

Bachelorthesis

Musik machen mit intuitiven Interfaces

Making music using intuitive Interfaces

Fabian Seipel

2011



Bachelorthesis

Musik machen mit intuitiven Interfaces

Making music using intuitive Interfaces

Fabian Seipel

Tag der Abgabe:

31. Dezember 2010

Betreuende(r) Mitarbeiter/-in:

Dr.-Ing. Marlene Helfert

Dipl.-Ing. Benjamin Franz

Erklärung zur vorliegenden Arbeit gemäß § 22/7 bzw. § 23/7 APB

Hiermit versichere ich, die vorliegende Bachelor Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Name: ………………………………………

Vorname: ………………………………………

Ort: ………………………………………

Datum: ………………………………………

Unterschrift: ………………………………………

Geheimhaltungserklärung / Rechte an der Arbeit

Hiermit verpflichte ich mich, bei Zusammenarbeit mit einem externen Auftraggeber, über alle mir im Rahmen der Bearbeitung meiner Bachelor Thesis im Zusammenhang mit dem Auftraggeber zur Kenntnis gekommenen Informationen Verschwiegenheit gegenüber jedermann zu bewahren. Die Veröffentlichung oder Weitergabe meiner Studien- bzw. Diplomarbeit darf in diesem Falle nur in Absprache mit dem Leiter des Instituts für Arbeitswissenschaft der TU Darmstadt erfolgen.

Bei Arbeiten ohne externen Auftraggebern liegen alle Rechte an der Studien- bzw. Diplomarbeit, darunter das Recht der Veröffentlichung, der Verbreitung und das Besitzrecht an der Idee zu gleichen Teilen bei IAD, Betreuer und dem/der Bearbeiter/in und können - innerhalb einer Frist von 2 Jahren nach Abgabe der Arbeit - nur durch Zustimmung von mindestens zwei der drei Parteien wahrgenommen werden.

Name: ………………………………………

Vorname: ………………………………………

Ort: ………………………………………

Datum: ………………………………………

Unterschrift: ………………………………………

Zusammenfassung

In der vorliegenden Bachelor-Thesis wird der Entwicklungsprozess eines Interfaces zur intuitiven Musikproduktion dargestellt, das final in der Lage ist mit Hilfe von Infrarotsignalen, die mit der menschlichen Mechanik verbunden sind, und dem Wiimote-Videospielcontroller, Bewegungen direkt in die Klangerzeugung umzusetzen und somit Musik zu machen.

Zunächst werden dazu grundlegende, der Thematik entsprechende Termini beschrieben, die die Grundlage des weiteren Vorgehens bei der Konstruktion der Schnittstelle bilden. Insbesondere der zentrale Begriff des Interfaces wird hierbei sehr ausführlich erklärt, wozu ferner ein Exkurs in die Technikphilosophie bemüht wird. Um danach den Bezug zum musikalischen Aspekt herzustellen, ist ein Einblick in die neuere Entwicklungsgeschichte der digitalen Musikproduktion von Nöten, woraus letztlich mit den zuvor gegebenen Begriffserklärungen eine Definition des musikalischen Interfaces erfolgt, das von diversen Beispielen gestützt wird.

Der Hauptteil dieser Arbeit, der auf dem theoretischen Grundlagenteil aufbaut, besteht aus dem eigentlichen Design des „Wii Infrared MIDI“ genannten Interfaces. Mit Hilfe der Infrarotkamera einer Wiimote-Spielkonsolenfernbedienung werden Bewegungen- und Gesten erkannt, die von einem mit Infrarotleuchtdioden besetzten LED-Handschuh ausgeführt werden. Durch die Bluetooth-Datenübertragungstechnik werden die aufgenommenen Bewegungsinformationen anschließend an den Computer weitergeleitet. Anschließend werden diese mit der in der Programmiersprache C# in der Entwicklungsumgebung Microsoft Visual Express 2010 erstellten Interface-Software analysiert und identifiziert, die daraufhin mit Musical-Instrument-Digital-Interface-Steuerinformationen virtuelle Instrumente zur Musikproduktion steuert..

Hierzu wird zunächst auf den Konzeptionsvorgang eingegangen, der neben der Vorstellung der verwendeten Sensortechniken der Wiimote auch die MIDI- und Bluetooth-Schnittstellen vorstellt, die für die Übertragung der Daten an den PC sowie die Verbindung von Interface-Software und den Computer-Instrument zuständig sind. Weiterhin werden die Anforderungen an Funktion und Bedienoberfläche des Schnittstellen-Programms gestellt, dessen Erstellung in der Entwicklungsumgebung Visual Studio 2010 von Microsoft, die ebenfalls vorgestellt wird, anschließend detailliert geschildert wird. Abschließend wird für den Nutzer eine Installations- und Bedienungsanleitung formuliert, sodass jeder Anwender mit „Wii Infrared MIDI“ arbeiten kann.

Inhaltsverzeichnis

1. Einleitung

Mit der Bezeichnung „Ubiquitous Computing“ prägte WEISER (1991) in seinem Aufsatz „The Computer for the 21st Century“ einen Begriff, dessen Sachverhalt und Folgen für uns heutzutage im Alltag völlig selbstverständlich erscheint: die Allgegenwärtigkeit der computerbasierten Informationsverarbeitung. Laut DE.STATISTA.COM (2011) verwenden 60 Prozent aller Beschäftigten in Deutschland einen Personal Computer am Arbeitsplatz. Mit einem Blick auf die private Nutzung des PCs wird diese Omnipräsenz sogar noch deutlicher: Der Prozentsatz der deutschen Haushalte, die einen Computer besitzen, liegt bei 82 Prozent (vgl. BITKOM.ORG 2011). In beiden Fällen ist darüber hinaus auch noch eine steigende Tendenz zu vermerken. Der PC, dem 1943 der damalige IBM-Chef Thomas J. Watson ein ähnliches Erfolgspotential versprach wie Kaiser Wilhelm II. dem Auto, ist wahrscheinlich die momentan bedeutsamste Maschine in fast allen Zweigen der heutigen Gesellschaftsform. Auch in der Musik und vor allem in der Musikproduktion nimmt dessen Bedeutsamkeit immer weiter zu:

> „Today Computers are ubiquitous in music. There is almost no recorded music that does not involve the use of a computer somehow or other, and the ever decreasing cost of the technology means that a bona fide home computer music studio is within the means of any erstwhile member of the middle class of the western world”. (OSTERTAG 1996)

Es ist mittlerweile also die absolute Ausnahme, dass ein produziertes Musikstück ohne den Einsatz von Computern auskommt, sei es die Aufnahme von Tonmaterial, deren Bearbeitung, Mischung oder gar das eigentliche Instrumentenspiel in jedweder Form, das am PC mit den modernen Mitteln der Soft- und Hardwaretechnik bewerkstelligt werden kann. Das mag, vor allem dem klassischen Instrumentalisten seltsam vorkommen, da jedes übliche Instrument ein individuelles, mechanisch anzuregendes Eingabemedium besitzt, dass vom Musikvirtuosen in der richtigen Form und in einer bestimmten zeitlichen Folge mit einer jeweiligen Art Impuls betätigt werden muss. Dies beschreibt beispielsweise die Seiten, die bei der Geige mit einem Bogen, bei der Gitarre mit den Händen oder beim Klavier mittels der Hammermechanik über die Tasten in Schwingung versetzt werden, oder kann die Variation des Luftdrucks sein, die bei den meisten Blechblasinstrumenten durch den Mund bewerkstelligt wird. Da jedoch die typischen Eingabemedien für den Computer aus Maus und Tastatur bestehen, die vor allem zum Spiel eines Instruments wenig geeignet sind, liegt es nahe, neue Schnittstellen, sogenannte Interfaces zu entwickeln, die für die Musikproduktion am PC wesentlich geeigneter sind. Dieser Form der Mensch-Maschine-Schnittstellen, die vor allem in Kapitel 3.2 genau definiert und im Bereich der Musikproduktion untersucht und detailliert erklärt werden, widmet sich indessen eine ganzer Wirtschaftszweig, denn mit der Digitalisierung und Virtualisierung (siehe Kapitel 3.1) der Erzeugung und Bearbeitung von Musik, die mittels des PCs möglich wurde, ist diese für fast jedermann erschwinglich geworden und somit ein großer Markt geworden (vgl. SMUDITS 2008, 257).

Im Rahmen dieser Bachelor-Arbeit sollen jedoch keine marktwissenschaftlichen Aspekte im Vordergrund stehen. Vielmehr ist es die Aufgabe ein komplettes, funktionsfähiges Interface-System für den World Usability Day 2011 zu entwickeln, das sich vor allem durch ein intuitives Bedienen auszeichnet und auch den Nutzer

anspricht, der vom Spiel eines Instrumenten und des weiteren von Notenschrift keine Kenntnis besitzt, also insgesamt wenig musikalische Grundkenntnisse vorweisen kann. Dabei ist vor allem die kreative Umsetzung musikalischer Ideen essentiell. Die fertig konstruierte Schnittstelle beinhaltet somit sowohl Hard- als auch Softwareumfang. Letzteres wird mit der Programmiersprache C# mittels Visual Studio Express 2010 codiert. Näheres dazu findet sich im Kapitel 7.1.

Die schriftliche Ausarbeitung gliedert sich nach der Einleitung in weitere fünf Teile. Zunächst werden die Begriffe intuitiv, Usability und Interface, die alle Kernaspekte der Themenstellung sind, beschrieben. Besonders die zwei letztgenannten Termini bedürfen einer etwas ausführlicheren und komplexeren Erklärung, um ihre Bedeutung vollständig zu klären. Daraufhin folgt ein kleiner Einblick in die Entwicklung der Rolle des Interfaces in der Musikproduktion, wozu zunächst der immer noch anhaltende Wandel dieser im momentanen Zeitalter der Digitalisierung dargelegt wird, damit daraufhin die Folgen für Interfacelösungen in der digitalen Produktion von Musik gezeigt werden können.

Als Nächstes folgen verschiedene technische Grundlagen, die zur Funktion des Systems benötigt werden und daher erläutert werden müssen. Dazu zählen neben der Wiimote die verwendeten MIDI- und Bluetooth-Interfaceprotokolle und die eingesetzte Musiksoftware, die schließlich als Klangerzeuger fungiert.

In Kapitel fünf wird anschließend, auf den beschriebenen theoretischen Grundlagen zu Interfacedesign und digitaler Musikproduktion, ein Konzept der Interfaces entwickelt, das sowohl Anforderungen an die grafische Benutzeroberfläche als auch an die verschiedenen Funktionen des Systems stellt.

Den Abschluss dieser Arbeit beschreibt das eigentliche Design des gesamten Interfaces. Hierzu zählen zum einen die Konstruktion der LED-Handschuhe, mit denen die Eingabe erfolgt und zum anderen die ausführliche Beschreibung der Softwareoberfläche sowie der Implementierung des „Wii Infrared MIDI“ Programms mit der gesamten Softwarearchitektur. Zuletzt wird eine Installations- und Bedienungsanleitung gegeben, die dem Nutzer auch ohne Lektüre dieser Arbeit die Verwendung des Interfaces ermöglicht.

Abschließend wird ein Ausblick auf zukünftig realisierbare Möglichkeiten mit ähnlichen Zielen, also der intuitiven digitalen Musikproduktion, gegeben.

2. Begriffe

Um im weiteren Verlauf immer unmissverständlich auf einige wichtige und in dieser Arbeit oft verwendete Vokabeln zurückgreifen zu können, die besonders relevant sind und teilweise auch den Titel dieser Arbeit charakterisieren, besteht dieses Kapitel vorrangig aus Erklärungen, die versuchen, die soeben genannten Begrifflichkeiten zu verdeutlichen. Dazu wird manchmal nur auf die wörtlich richtige Bedeutung verwiesen, in einigen Fällen, respektive der Komplexität der Termini, wird aber auch auf deren Gesamtheit zur Begriffsbildung näher eingegangen. Unter anderem ist auch der Sinn des World Usability Days, für den das System entwickelt wird, ein Thema.

2.1. Intuitiv

Ein in Verbindung mit Mensch-Maschine-Schnittstellen oft und gern verwendetes Adjektiv ist intuitiv. Die Phrase „intuitive Bedienung“ ist wohl eine der meistgebrauchten Redensarten in der Werbeindustrie wenn es um positive Eigenschaften der Nutzung eines Produktes geht. Auch das zu entwickelnde Interface in dieser Arbeit soll es verdienen damit charakterisiert zu werden. Darum erscheint es in diesem Kontext wichtig, dieses Fremdwort einmal genau zu betrachten.

Intuitiv beruht auf dem Nomen Intuition, das, abgeleitet von dem lateinischen Wort „intueri“ , soviel bedeutet wie „betrachten“, „erwägen“, im ursprünglichen Sinn „angeschaut werden“. Es beschreibt die Fähigkeit, Einsichten in Sachverhalte, Sichtweisen, Gesetzmäßigkeiten oder die subjektive Stimmigkeit von Entscheidungen ohne diskursiven Gebrauch des Verstandes, also ohne bewusste Schlussfolgerungen, zu erlangen. Es ist ein Teil kreativer unbewusster Entwicklungen, der vom Intellekt begleitet und überprüft wird (vgl. WIKIPEDIA, 2011). Der DUDEN (2007) schildert Intuition zum einen als „das unmittelbare, nicht diskursive, nicht auf Reflexion beruhende Erkennen, Erfassen eines Sachverhalts oder eines komplizierten Vorgangs“.

2.2. Usability

Einer der wirklich vielschichtigen Begriffe in dieser Arbeit ist Usability. Darin „finden sich die englischen Worte „use“ (gebrauchen, verwenden, benutzen) und „utility“ (der Nutzen, die Nützlichkeit)“ (vgl. STAPELKAMP 2010), aber auch das englische Wort ability, das im Deutschen so viel wie „Befähigung“ oder „Fähigkeit“ bedeutet. Es geht also einerseits um die Kenntnis und die Befähigung etwas zu gebrauchen und gleichzeitig um den Nutzen und somit den Vorteil, der mit dieser Nutzung einhergeht. Usability wird zudem „mittlerweile im deutschen Sprachgebrauch synonym zu den Begriffen „Benutzerfreundlichkeit/ Gebrauchstauglichkeit“ verwendet“ (vgl. BRUDER et al. 2010). Darum wird versucht diese beiden Ausdrücke im Folgenden auch hinreichend zu erklären.

Gebrauchstauglichkeit ist ebenfalls nach BRUDER (2010) laut der Norm DIN EN ISO 9241, 11 mit dem Titel „Ergonomie der Mensch-System-Interaktion“ definiert als „das Ausmaß, in dem ein Produkt durch bestimmte Nutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen“.

> „Effektivität bezeichnet [dabei] die Akkuratheit und Vollständigkeit, mit denen ein Ziel erreicht werden kann. Die Effizient beschreibt das Verhältnis vom Aufwand der Ressourcen zum Nutzen, die in Bezug auf die Zielerreichung notwendig sind. […] Zufriedenheit dagegen ist eine subjektive Komponente, die die Abwesenheit von Frustration, aber auch positive Einstellungen gegenüber dem Produkt beschreibt“ (BRUDER et al. 2010)

Nach dieser Definition der Norm ist es somit Voraussetzung zunächst den angesprochenen Kontext der Nutzung und ein gewisses Bild des Nutzers, etwa mit einem Modell, festzulegen. Dies betrifft seine Ziele und Aufgaben sowie sein gesamtes Umfeld. Abbildung 2.1 veranschaulicht diese Vorgehensweise bei der Produktentwicklung nach Usability-Prinzipien.

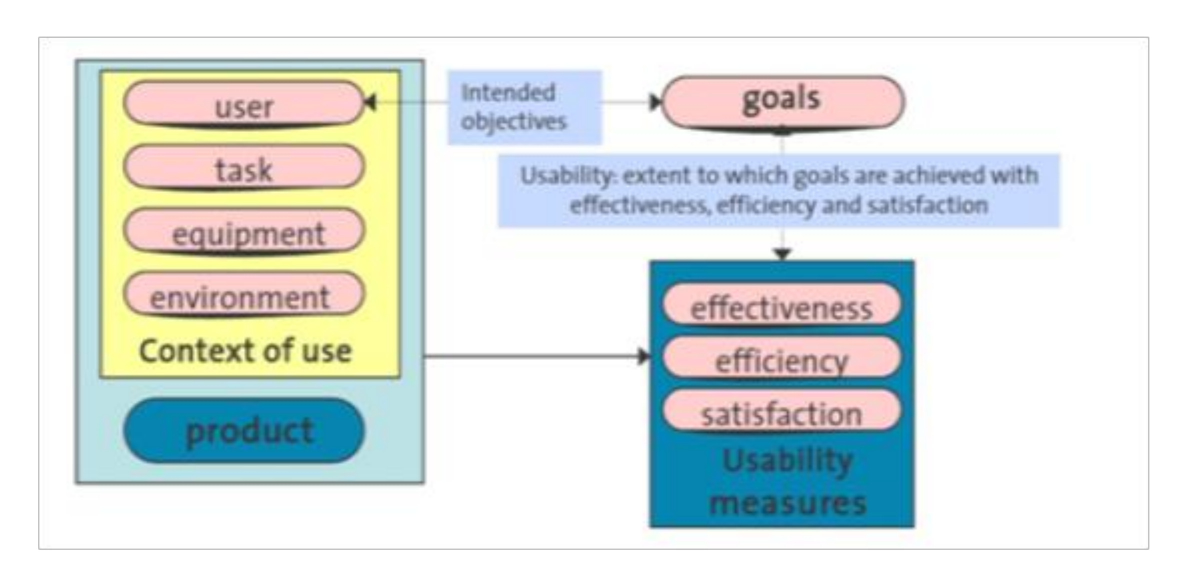


Abb. 2.1: Produktentwicklungsprozesses nach Usability-Prinzipien

Usability lässt sich weiterhin auch immer nur bei einer Kommunikation zwischen Mensch und Maschine definieren, wenn also eine Interaktion dieser Parteien stattfindet und Informationen ausgetauscht werden müssen (vgl. STAPELKAMP 2010). Ist nun der erwähnte Kontext festgelegt kann untersucht werden, inwieweit ein System effiziente und effektive Arbeit gewährleistet und den Anwender seine Ziele zufriedenstellend erfüllen lässt. Zur Evaluation eines Produktes hinsichtlich dieser Attribute gibt es mittlerweile etliche Verfahren. Diese zu thematisieren ist allerdings nicht Aufgabe der Arbeit.

Den zweiten Terminus zu Usability beschreibt die Benutzerfreundlichkeit, unter der man im Allgemeinen die Nutzungsqualität bei einer Interaktion mit einem System versteht. BRUDER et al. (2010) geben mit SPINAS et al. (1990) etwa folgende Definition:

„…ein Dialogsystem ist dann als benutzerfreundlich zu bezeichnen, wenn es den Benutzer durch vielfältige Anwendungsmöglichkeiten von Routinearbeit entlastet und ihn – bei hoher Verfügbarkeit – in der Interaktion am Bildschirm seiner Erfahrung und Geübtheit angemessene Freiheitsgrade für unterschiedliche Vorgehensweisen gewährt, ohne ihm dadurch neue Routinearbeit und komplizierte Bedienungsoperationen aufzubürden.“

Beigefügte Abbildung 2.2, ebenfalls aus SPIAS et al. (1990), verdeutlicht dies.

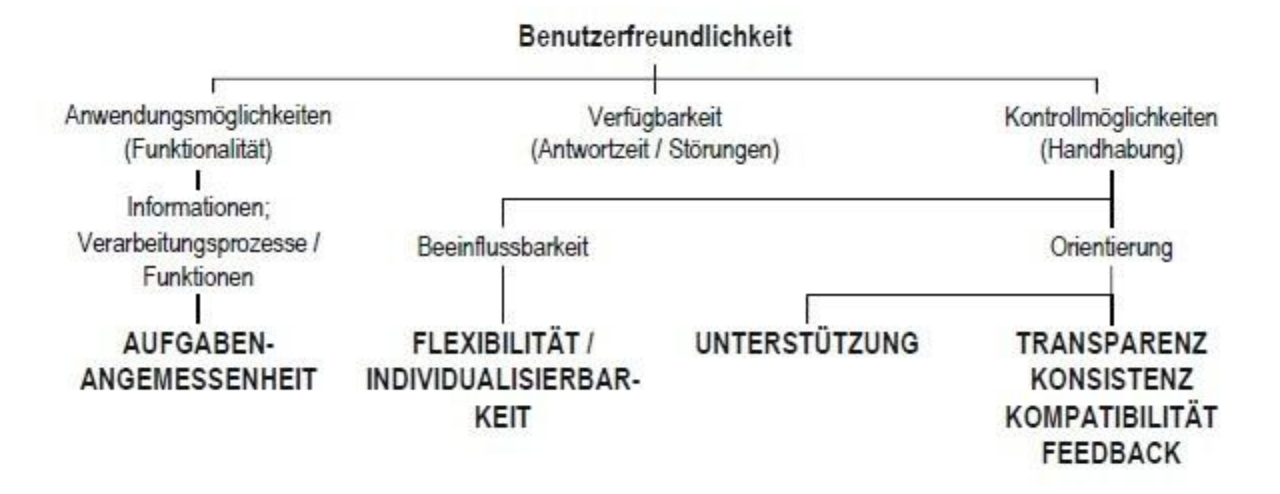


Abb. 2.2: Aspekte der Benutzerfreundlichkeit

Im Gegensatz zur Gebrauchstauglichkeit enthält der zweite Aspekt der Usability aber auch eine wesentlich subjektivere Komponente. Während Erstere vielleicht so etwas wie eine Art Mindestanforderung darstellt, beinhaltet Nutzerfreundlichkeit ebenfalls emotionale Aspekte und lässt sich daher auch zu den Begriffen Joy of Use und User Experience, die in Kapitel 2.3.3.2 näher betrachtet werden, nicht abgrenzen. Eine Eingrenzung in feste Normen und Richtlinien lässt sich kaum vereinbaren. Jeder Einzelne, das wird im Alltag schnell deutlich, erfährt die Anwendung eines Produktes auf seine eigene Art und Weise. Jeder besitzt andere Grundkenntnisse und hat spezielle Anforderungen an das System. Mit der Zeit und dem Fortschritt der technischen und sozialen Entwicklung ändern sich diese Ansprüche natürlich auch. Insgesamt tragen zu einer nutzerfreundlichen Gestaltung viele verschiedene Faktoren bei. Grundlegend ist die einfache, selbsterklärende Bedienung. Produkte, die ihre Bedienungsanleitung überflüssig erscheinen lassen, da Elemente der Bedienung bekannt sind oder ihre Aufgabe vom Anwender ohne Hilfe erkannt werden kann, sind hier gemeint. In einem Video-Interview mit dem Ehrenvorsitzenden des World Usability Days 2010, John Hockenberry, lässt sich unter anderem eine Frau in Bezug auf die einfache Handhabung ihres Handys mit folgenden Worten zitieren: „I don't have to feel stupid about using technology“ (vgl. WORLDUSABILITYDAY.ORG, 2011). Dieser Ausspruch beschreibt genau das Empfinden, das nutzerfreundliches Design ausmacht. Vor allem Menschen, die sich nicht als technikaffin beschreiben würden, oft trifft das auf ältere Mitglieder unserer Gesellschaft zu und/oder Personen die kein Interesse haben, den rasanten technologischen Fortschritt zu verfolgen, sollen mit einer Produktgestaltung nach dem Usability-Prinzip keine Probleme bei der Bedienung von systemischen

Schnittstellen haben. Oft mangelt es hier jedoch an Rücksicht auf diese Menschen, die sich mit unserer technisierten Informationsgesellschaft weniger identifizieren können. Gleiches trifft auch auf behinderte oder in anderer Form beeinträchtigte Personen zu. Hierbei wird die sogenannte Barrierefreiheit von Produkten gefordert. Diese „bezieht sich auf die Forderung, dass ausnahmslos alle Menschen, auch solche mit motorischen, perzeptiven, kognitiven, sprachlichen oder altersbedingten Einschränkungen ein bestimmtes Produkt benutzen können" (BRUDER et al. 2010). Der Begriff der Barrierefreiheit ist sogar in §4 des Gesetzes zur Gleichstellung behinderter Menschen (BGB) festgeschrieben.

Für die Art der Dialoggestaltung, auf die sich auch die obige Definition nach SPIAS et al. (1990) bezieht, wurden zwar keine Gesetzte verfasst, jedoch aber Normen für nutzerfreundliche Gestaltung formuliert. Ein Dialog ist dabei, mit KÖNIG und RÖBIG (2010) nach DIN 66234, „ein Ablauf, bei dem der Benutzer zur Abwicklung einer Arbeitsaufgabe – in einem oder mehreren Dialogschritten – Daten eingibt und jeweils Rückmeldung über die Verarbeitung dieser Daten erhält". Das kann prinzipiell jegliches Mensch-Maschine-System sein, ist aber vor allem für die Gestaltung von interaktiven Systemen solcher Art, wie Webseiten oder Software, besonders interessant, da, wie in der Einleitung bereits dargelegt, der Personal Computer und somit auch die darauf benutzten Programme und das Internet in der heutigen Arbeitswelt wahrscheinlich den größten Teil solcher Dialogsystemen darstellen und diese verwenden. In DIN EN ISO 9241-110 werden deswegen die Grundsätze dieser Dialoggestaltung definiert:

- **Aufgabenangemessenheit**: geeignete Funktionalität, Minimierung unnötiger Interaktionen
- **Selbstbeschreibungsfähigkeit**: Verständlichkeit durch Hilfen / Rückmeldungen
- **Lernförderlichkeit:** Anleitung des Benutzers, Verwendung geeigneter Metaphern
- **Steuerbarkeit:** Steuerung des Dialogs durch den Benutzer
- **Erwartungskonformität:** Konsistenz, Anpassung an das Benutzermodell
- **Individualisierbarkeit:** Anpassbarkeit an Benutzer und an seinen Arbeitskontext
- **Fehlertoleranz:** Die Funktionsweise des Systems auch bei unvorhergesehenen Fehlern aufrechterhalten

Auch der Prozess der nutzerorientierten Gestaltung, den Abbildung 2.3 zeigt, ist in DIN EN ISO 9241 festgelegt. Er besteht aus vier Teilschritten. Zuerst wird der gesamte Nutzungskontext festgelegt um daraus im zweiten Schritt die Anforderungen zu spezifizieren. Daraufhin kann eine Lösung, meist in Form eines Prototyps produziert werden, die abschließend hinsichtlich der formulierten Anforderungen bewertet wird. (vgl. RÖBIG und WAGNER, 2010)

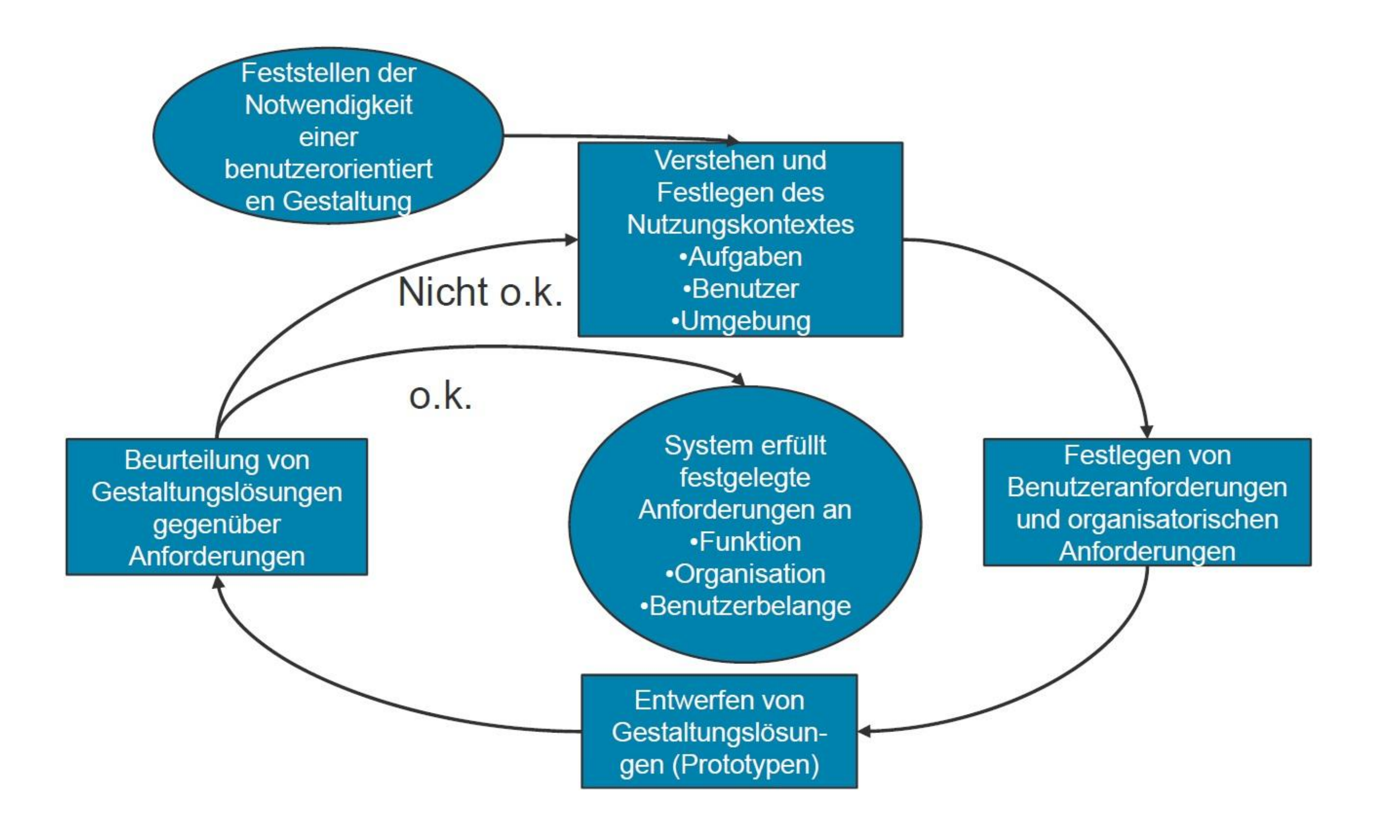


Abb. 2.3: Prozess nutzerorientierten Gestaltung

Unter dem Begriff Usability, so lässt sich schließlich zusammenfassen, wird weitaus mehr verstanden als nur die Steigerung von Effizienz und Effektivität im Umgang mit einem Produkt. Subjektive Faktoren wie die Freude ein System zu benutzen treten immer mehr in den Vordergrund. Dem Anwender soll die Interaktion einfach gemacht werden, der Anwendungsprozess soll ihn bei der vollständigen Erfüllung seiner Aufgabe unterstützen und das mit möglichst wenig Aufwand in jeglicher Art. Fehler des Benutzers sollen dabei erkannt werden und er darauf hingewiesen werden. Der Nutzer muss erkennen welche Auswirkungen seine Handlungsschritte haben, die gesamte Bedienung muss möglichst einfach gestaltet werden und auch etwaige Individualisierungen sollen möglich sein um bestenfalls zur Optimierung des Systems beizutragen. Bei der Gestaltung von Interfaces werden, wenn möglich, alle diese Eigenschafte, die sich unter der Definition von Usability, verbergen, berücksichtigt.

Die Relevanz von Usability heutzutage spiegelt sich schon allein darin wieder, dass es eine eigene weltweite Veranstaltung zu diesem Thema gibt: den World Usability Day, der im englischen auch „make things easier day“ heißt und seit dem Jahr 2005 in mittlerweile mehr als 43 Ländern unter dem Motto „Making Life Easy“ veranstaltet wird. Für Firmen und Studenten geht es um die Präsentation von neuen Produkten zur Vereinfachung des täglichen Alltags und den Austausch von Ideen zu den Themen Benutzerfreundlichkeit und Gebrauchstauglichkeit. Auch Vorträge von Unternehmenskräften oder Mitarbeiter der Universitäten gehören zum Programm (vgl. WORLDUSABILITYDAY.ORG, 2011). Das Interface, das in dieser Arbeit erstellt werden soll, wird auf dem nächsten World Usability Day in Darmstadt im Jahr 2011 vorgestellt.

2.3. Interface

Um den Sinn von Schnittstellen zwischen Mensch und Maschine aufzuzeigen, wird zunächst kurz eine kleine Einleitung in die Technikphilosophie gegeben um daraus und aus dieser Sicht die Notwendigkeit solcher Interfaces zu verstehen. Daraufhin folgt eine nicht-philosophische, allgemeine und eher praktische Definition dazu, die auch die Kunst des Designs solcher Interfaces beleuchtet.

2.3.1. Technikphilosophische Betrachtung

Als einleitenden Gedanken in dieses Kapitel wird zunächst auch kurz auf den Sinn von Mensch-Maschine-Schnittstellen aus technikphilosophischer Sicht eingegangen, der zur allgemeinen Definition des Interface-Begriffs hilfreich ist. Dazu wird zu Anfang, ebenfalls aus dieser Sichtweise, die Beziehung und Entwicklung von Technik zum Menschen dargestellt und daraus dann der Begriff der Transklassischen Technik entwickelt.

2.3.1.1. Mensch-Technik-Relation

Die Menschheit hat im Laufe ihrer Geschichte schon immer Technikformen entworfen um sich weiterzuentwickeln. Dort, wo die dem Menschen, natürlich gegebenen Mittel nicht mehr ausreichen oder nie vorhanden waren, bedient er sich ihrer Möglichkeiten, um neue Ziele zu erreichen und den materiellen Wohlstand zu sichern. Der Technikphilosoph Arnold Gehlen spricht hierbei vom Menschen als Mängelwesen, der die Technik als Ersatz, Entlastung und Verstärkung seiner gottgegebenen Organe sieht. (vgl. GEHLEN 1957)

Zu Beginn dieser technischen Evolution steht die Erfindung von Werkzeugen, die den homo sapiens grundsätzlich vom Tier abgrenzt. Inzwischen haben sich die, vor allem in den letzten zwei Jahrhunderten, von Menschenhand konstruierten Maschinen immer mehr verkompliziert und spezialisiert. Dabei werden hier bewusst die Begriffe Werkzeug und Maschine verwendet, die deutlich zu trennen sind und deren Unterschied auch in Bezug auf die Frage der Verantwortung hinsichtlich ihrer Benutzung kenntlich gemacht werden soll. Denn während nach handlungstheoretischen Gesichtspunkten beim klassischen Werkzeug noch eine direkte Kontrolle und Möglichkeit der Korrektur und Einflussnahme besteht, kann bei der Nutzung einer Maschine meistens nur noch eine Steuerung oder Regelung erfolgen und anschließend das Ergebnis überprüft werden. Bei der Verwendung des Hammers zum Beispiel ist der Anwender alleinig verantwortlich für das Ergebnis seines Einsatzes. Er führt die Bewegung, die zum Schlag mit dem Hammer führt, selbst aus, sieht das Ergebnis und kann durch Reflexion darüber in weiteren Arbeitsschritten Korrekturen vornehmen. Beim Gebrauch einer Maschine jedoch, ist die Aufgabe des Menschen meist darauf beschränkt, ihre Bedienung vorzunehmen und nachdem die Maschine ihren Arbeitsschritt ausgeführt hat, die Konsequenz dessen zu kontrollieren. Was genau in dem erwähnten Arbeitsablauf allerdings passiert und wie dies geschieht, weiß der Nutzer nur in den wenigsten Fällen, da die Mehrzahl der heute verwendeten Maschinen dafür eine viel zu hohe Komplexität aufweisen. Das Automobil beispielsweise ist eines der wichtigsten Maschinen heute um individuelle Mobilität zu gewährleisten. Dem Großteil unserer Gesellschaft allerdings reicht das Wissen mit

dem seine Benutzung möglich wird und verzichtet dabei auf die präzise Kenntnis des Zusammenspiels der etlichen technischen Elemente, die für die Gesamtfunktion des Autos zuständig sind. Aus diesem Grund kann der Nutzer auch nicht mehr alleinig für die Folgen seines Handelns verantwortlich gemacht werden, da der Entwickler der Maschine ebenfalls bei ihrem Gebrauch einen Teil der Verantwortung trägt, obwohl er daran nur indirekt und zwar durch die Konstruktion beteiligt ist. Nach Unfällen bei Fehlverhalten von Maschinen ist es daher nicht mehr eindeutig möglich die Schuldfrage zu klären, da sich keine klare Trennung der Verantwortung mehr vollziehen lässt. (vgl. HUBIG und JELDEN 1994)

In der Weiterführung dieser Einteilung von Technik stellen HUBIG und JELDEN (1994) den Ausdruck „System“ als Weiterentwicklung der Maschine vor. Bei diesen großtechnischen Systemen handelt es sich etwa um die Elektrizitätsversorgung, das Verkehrssystem oder Funk und Fernsehen. Dieser Art von Technik, die meist von Organisationen wie beispielsweise Großfirmen oder dem Staat betrieben werden, kann man sich als Einzelperson nicht mehr problemlos entziehen. Viele oder fast alle gebräuchlichen Gerätschaften sind auf diese Infrastruktur der großtechnischen Systeme angewiesen. Sie sind essentiell für das Alltagsleben und es besteht darum also ein gewisser Zwang sie zu nutzen. „Das heißt, eine Unzahl unserer Handlungsvollzüge ist institutionell oder organisatorisch mit derartigen Systemen verwoben, die – und das ist entscheidend – eine Art Automatismus entfalten, indem die uns zu bestimmte Verfahrensweisen zwingen“ (HUBIG und JELDEN 1994). Dem Individuum ist es somit zwar möglich die beschriebenen Systeme zu nutzen, allerdings nur noch in vorgegebenen Handlungsspielräumen ohne oder mit eingeschränkten Optionen, womit das unabhängige Vorgehen mittels Steuerung bzw. Regelung durch den Einzelnen nicht mehr möglich ist und damit auch keine Übernahme von Verantwortung mehr.

2.3.1.2. Transklassische Technik

HUBIG entwickelt in Bezug auf derartige Ausprägungen des technischen Fortschritts den Begriff der transklassichen Technik, der prinzipiell das Phänomen beschreibt, dass sich Natur und Technik in manchen Disziplinen der Wissenschaft nicht mehr eindeutig trennen lassen. Dabei ist die Veränderung der Schnittstellen zwischen menschlichem Akteur und technischem System von besonderer Bedeutung. Diese Schnittstelle, oder eben auch Interface, wie es im englischen genannt wird, beschreibt den Punkt an dem der Austausch von Information der beiden Partizipierenden stattfindet. Kann diese bei Systemen transklassischer Technik jedoch nicht mehr eindeutig definiert und wahrgenommen werden, so verschwindet die Fähigkeit sich dazu in Bezug zu setzen, was Grundvoraussetzung dafür ist den technischen Fortschritt immer wieder zu überdenken. LUHMANN spricht in diesem Fall auch von fehlender Medialität, an dessen Widerstand die Herausbildung von Kompetenz stattfinden kann. Nach HUBIG geht damit sogar die Möglichkeit des Abduzierens, also der Bildung von Hypothesen aufgrund von kenntniserweiternden logischen Schlüssen, verloren, auf die jedoch wissenschaftliches Arbeiten aufbaut. Es fehlt eine Herausbildung von Erfahrung, die als Differenz zwischen vorgestelltem und realisiertem Zweck eine große Rolle spielt.

Exempel solcher Technikausprägungen sind etwa die Nanotechnologie oder die rechnerunterstützte Informations- und Datenverarbeitung. Angeführt werden können hier spezielle Beispiele sogenannter Biofakte

wie die von Donna Haraway beschriebene Krebsmaus, die zum einen Labortier ist und durch die Einpflanzung eines menschlichen Krebsgens die Natur des Menschen vertritt, zum anderen aber auch gleichzeitig technische Züge als reproduzierbares Patent besitzt (vgl. NORDMANN). Ähnlich verhält es sich mit Informatisierung und Virtualisierung der Handlungsumwelt. Dies beschreibt eine Vermischung von realer und virtueller Welt, die dann besonders problematisch erscheint, wenn die Unterscheidung derer nicht mehr klar begriffen werden kann und das Gesamtsystem keinen Zugang zur Realität mehr besitzt. (vgl. HUBIG) Diese auch „mixed realities“ genannten Zustände können Simulatoren sein oder auch einfach simple Navigationsgeräte. Extreme Ausprägungen einer solchen Weiterentwicklung zeigen Filme wie die Matrix Trilogie auf, in denen sich fast die komplette Menschheit in einer virtuellen Realität befindet ohne sich dessen bewusst zu sein.

2.3.2. Begriff des Interface

Der philosophische Exkurs in den vorigen Kapitel soll dabei zeigen, wie wichtig die Definition und die klare Festlegung von Schnittstellen sein kann, um dadurch genügend Transparenz auch beispielsweise in sehr komplexen transklassichen Systemen zu schaffen, damit weiterhin die Möglichkeit besteht einen klaren Bezug zu diesen System herzustellen zu können. Natürlich sind dabei viele Gedanken weit über die heutige Gestalt von Maschinen hinaus fortgeführt. Es ist jedoch aber durchaus schon Technik vorhanden, die klare Tendenzen und Merkmale dafür aufweist. Erwähnt seien die bereits beschriebenen großtechnischen Systeme wie beispielsweise die Elektrizitätsversorgung.

Der Fachbegriff Interface stammt ursprünglich aus dem Lateinischen und setzt sich aus „inter“ (lat.: dazwischen) und „agere“ (lat.: handeln, machen) zusammen. Es beinhaltet ebenso das englische Wort „face“, zu Deutsch „Gesicht“. Demnach weist Interface der wörtlichen Bedeutung nach auf eine Art Handlung zwischen etwas hin, beschreibt aber auch gleichzeitig eine spezifische Oberfläche, die vermitteln kann (vgl. CLAUSSEN 2006). Zusätzlich steht es in direktem Bezug zur Interaktion, dem Nomen zu interagieren, einem Fremdwort, dass laut DUDEN so viel wie „wechselseitig in seinem Verhalten beeinflussen“ bedeutet. Das Interface ermöglicht demzufolge den Dialog und den Austausch von Informationen zwischen zwei Parteien mittels einer Schnittstelle. Dabei kann dieser Austausch zwischen Mensch und Maschine ablaufen oder aber die Technik als Medium fungieren um die Verständigung zwischen zwei Individuen zu ermöglichen. Prinzipiell ist aber jede Struktur, die eine Kommunikation ermöglicht eine Art Interface. STAPELKAMP (2010) beschreibt das Interface auch als die Voraussetzung für Interaktion. Es ist dem Nutzer also ein Werkzeug um seine Anweisung dem technischen System mitzuteilen. Gleichzeitig soll es in diesem Fall natürlich ebenfalls die Vermittlung der Rückgabeinformation der Maschine ermöglichen. Dabei müssen Input des Nutzers und Output des Systems Formen annehmen, die beiden Seiten gleichermaßen gerecht werden und verstehen können (vgl. CLAUSSEN 2006). In manchen Fällen besteht heute das zu bedienende Produkt sogar fast nur noch aus dem eigentlichen Interface. Im Falle eines Laptops oder Handys ist die Oberfläche alleinig mit Bedienelementen oder Ausgabebildschirmen ausgestattet. Bei Touchscreen-Interfacelösungen wird sogar dieselbe Fläche als Ein- und Ausgabemedium verwendet.

2.3.3. Interfacedesign

Verantwortlich für das Entwerfen dieser Medien ist dabei das sogenannte Interfacedesign. Unter diesen Begriff versteckt sich jedoch viel mehr als nur die Gestaltung von Oberflächen um Interaktion zu ermöglichen. In vielen Firmen gibt es eigene Abteilungen, die sich mit diesem Aufgabenfeld befassen, da die Gestaltung der Schnittstellen einen immer größeren Stellenwert für sich beansprucht, nicht zuletzt, da viele technische Konsumprodukte auch immer kleiner werden. Als Vorzeigebeispiel dient hier etwa die neue Spezies der Tablet-PCs, die eine Mischung aus Laptop und Handy darstellen. Abbildung 2.4 zeigt das IPad 2, ein Beispiel dieser Technikspezies der Firma Apple. Es wird bei der Konzeption von Schnittstellen zusätzlich noch unterteilt zwischen dem Interactiondesign, das sich mehr auf das Verhalten im Nutzungskontext des Interfaces konzentriert und dem eigentlichen Interfacedesign, das eher die Gestaltung und Form des Objekts betrifft. Da die beiden Sparten sich jedoch kaum mehr trennen lassen, werden beide Themen oft unter dem Begriff Interfacedesign zusammengefasst zu dem auch Software- und Websitedesign zählen. (vgl. STAPELKAMP 2010)



Abb. 2.4: Apple IPad 2

2.3.3.1. Eingabe- und Ausgabemedien

Das noch recht junge Fachgebiet hat sich nicht zuletzt deswegen entwickelt, da vor allem die Computertechnologie einen immer größeren Einfluss auf unsere Gesellschaft in beruflicher und persönlicher Sicht vereinnahmt und um die steigende Komplexität dieser Systeme benutzergerecht zu verpacken und deren Bedienung weiterhin problemlos zu ermöglichen, müssen sinnvolle Interfaces entwickelt werden. Die Möglichkeiten der Kommunikation sind zwar an die Mittel des Menschen gebunden, Information bereitzustellen wie auch aufzunehmen und zu verarbeiten, trotzdem ist die Zahl der Wege dies zu gestalten unüberschaubar. Neben der visuellen Informationsrezeption des Menschen, die wohl noch zumeist angesprochen wird, sind auch taktile und auditive Wahrnehmung bei manchen Anwendungen gefragt. Welche Form im Einzelnen am besten verwendet wird, hängt vom gesamten Nutzungskontext ab und muss bei jeder einzelnen Entwicklung eines Interfaces neu überdacht werden. Oftmals ist eine Verbindung verschiedener Mittel von Vorteil. Kann per Bildschirm vielleicht die größte Menge an Informationen übertragen werden, so können Audio Signale beispielsweise bei Warnungen zweckdienlicher sein (vgl. KÖNIG 2010)

Auch für die klassischen Bedienelemente des Personal Computers wird in vielen Bereichen an Verbesserung geforscht. Mögen Maus und Tastatur zwar für die meisten Programme ausreichen, so gibt es jedoch auch Anwendungen, die mit anderen Formen der Eingabe wesentlich leichter und effektiver zu bedienen sind. Sie fallen unter den Begriff der Tangible Interfaces, die dem Nutzer ermöglichen sollen, unmittelbar mit digitalen Daten und Diensten zu interagieren. Tangible bezeichnet übersetzt etwa greifbar oder fühlbar. Vor allem im Bereich des kreativen Gestaltens wie beim Grafikdesign oder der Musikproduktion scheint diese Art der Eingabe zweckdienlicher. Auch bei der Entwicklung von Spielekonsolen zeichnet sich der Übergang vom normalen Gamepad-Controller zu neuen Formen der Eingabe ab. Alle führenden Hersteller auf diesem Gebiet bieten mittlerweile Konsolen an, die die Bewegungen des Spielers erkennen und verarbeiten können. Als Beispiel dient hier die so genannte Kinect-Erweiterung der Xbox 360 von Microsoft, die ebenfalls zu Steuerung von normalen PC's umgebaut werden kann. Ein Beispiel zeigt das Video auf KINECTHACKS.NET. Interfaces zur digitalen Musikproduktion werden in Kapitel 3.2 ausführlich vorgestellt. Anhand dieser Beispiele soll deutlich werden, dass wenn auch Maus, Tastatur und Bildschirm die weitaus meistgenutzten Ein- bzw. Ausgabegeräte für den PC darstellen, sie dennoch nicht die Einzigen und vor allem nicht immer die Effektivsten sind. Die Interfaceeinheit, die im Laufe dieser Arbeit entwickelt wird, fällt ebenfalls unter die Kategorie der kreativen Eingabemedien am Personal Computer.

2.3.3.2. User Experience und Joy of Use

Eine große Rolle bei der Entwicklung des Interfaces spielt auch der Begriff User Experience, zu Deutsch: Nutzer- oder Nutzungserlebnis. Dies soll all die Erfahrungen beschreiben, die der Anwender mit einem Produkt erlebt und ist daher auch immer mit einer subjektiven Erfahrung des Einzelnen verbunden, die auch als Joy of Use verstanden wird. Mit der Aussage „You cannot NOT have a user experience" zeigt Lou Carbone, dass Erfahrung durch Anwender prinzipiell immer stattfindet. Ziel einer Gestaltung nach derartigem Verständnis ist die Kreation eines ganzen Erlebnisses mit einem bestimmten Produkt und vor allem auch die Erfüllung der Erwartungen des Anwenders an das Produkt. Natürlich ist es hier von Vorteil, diese zu kennen bzw. herauszufinden, bevor der Entwicklungsprozess eines Interfaces beginnt. Insgesamt vereinen sich unter User Experience, neben den Prinzipien von Usability aus Kapitel 2.2, die dem Nutzer zusammenfassend ein effizientes, effektives und zufriedenstellendes Arbeiten ermöglichen sollen, auch zusätzlich noch der angesprochene Begriffe Joy of Use und nach Abbildung 2.5 auch der Look, also das äußerliche Erscheinungsbild, das den Konsumenten ansprechen soll. (vgl. STAPELKAMP 2010)

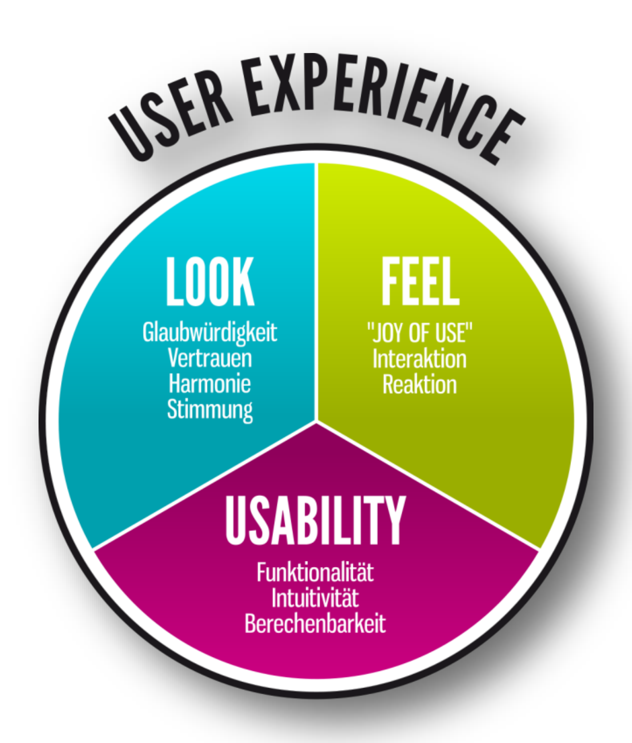


Abb. 2.5: User Experience

HATSCHER (2001) gibt dazu folgende Definition:

> „Joy of Use [...]ist das freudvoll-genussreiche Erleben der Qualität der Interaktion und der Möglichkeiten, die sich für einen bestimmten Nutzer in einem bestimmten Kontext als Folge des überwiegend unauffälligen, hervorragenden Funktionierens und aufgrund der den Nutzer ästhetisch ansprechenden Gestaltung durch motivierten und den Zielen und Interessen des Nutzers entsprechenden Gebrauch der Software manifestiert."

Dies muss dabei nicht immer mit den Richtlinien von Gebrauchstauglichkeit im Einklang sein. Auch komplexere Produkte, die dem Anwender in einer positiven Art herausfordern, ohne ihn gleichzeitig zu überfordern, sind Ergebnis von Gestaltung hinsichtlich Joy of Use. Für den Nutzer kann ebenfalls eine solche Auseinandersetzung die Freude am Gebrauch des Produkts darstellen. Computerspiele etwa sind meist alleinig darauf ausgelegt dem Anwender eine Herausforderung zu bieten. Im Allgemeinen stellt Joy of Use unter dem Begriff User Experience den wohl subjektivsten Teil des Nutzerempfindens dar. Darum ist es auch problematisch hier eine feste Systematik zum Vorgehen beim Design nach derartigen Strategien festzulegen (vgl. HASSENZAHL 2009). Das vielleicht bekannteste Beispiel für Joy of Use sind die Gebrauchsfunktionen des Multitouch-Interfaces des IPhones oder IPads der Firma Apple (siehe erneut Abbildung 2.4) Der Anwender kann sich durch Personalisierungen der medialen Umgebung vor allem durch die vielen verschiedenen Apps (Kurzform von Applikation) eine Erlebniswelt schaffen , die gefüllt ist mit Assoziationen. Es vermittelt den Eindruck als könne der Nutzer damit eine eigene, ganze Geschichte erleben. Apple baut sich mit dieser Art der Interpretation von Joy of Use gleichfalls eine komplette Firmenphilosophie und –identität auf. Hierbei wird oftmals auch von Erlebnisarchitektur gesprochen, wobei eine Dienstleistungen weit über Inhalte und Informationsverarbeitung hinaus geht (vgl. STAPELKAMP 2010). Donald Norman, ein bereits emeritierter Professor für Kognitionswissenschaften der University of San Diego, lässt sich abschließend zu User Experience mit folgenden Worten zitieren: “It’s not enough that we build products that function, that are understandable and usable, we also need to build products that bring joy and excitement, pleasure and fun, and yes, beauty to people’s lives.” (NORMAN 2004)

3. Entwicklung des Interfaces in der digitalen Musikproduktion

Folgende Kapitel dienen dazu ein kurzes Resümee über die Entstehung und Entwicklung der digitalen Musikproduktion und ihre Auswirkungen und Folgen zu geben. Anhand dieser Fakten und der soeben erfolgten Definition der Begriffe Interface bzw. des Interfacedesign, wirft Kapitel 3.2 danach einen ausführlichen Blick auf die Art musikalischer Interfaces, die zur Konzeption und Realisation von Musik am PC mittlerweile gebräuchlich ist. Darüber hinaus werden neuartige digitale Instrumente und Eingabemethoden für die intuitive digitale Musikproduktion vorgestellt, zu denen auch das in dieser Arbeit entwickelte System zählt.

3.1. Neuere Entwicklung der Musikproduktion

Um die sehr weitreichende und verzweigte Geschichte, die letztendlich zur Musikproduktion am Personal Computer führt in diesem Rahmen anschaulich zu beschreiben, wird hier nochmals unterteilt in die Bereiche der Digitalisierung und der Virtualisierung von Musikerzeugung, die sich, unterlegt mit Beispielen, auf Entstehung und Beschreibung der Fakten der jeweiligen Entwicklungen beschränkt und im Anschluss auf die musikalischen und gesellschaftlichen Konsequenzen, die daraus resultieren, eingeht.

3.1.1. Digitalisierung

Wie bei vielen, die Gesellschaft verändernden Innovationen, sind technische Neuerungen und Erfindungen der Anfang dieser musikalischen Evolution. In diesem Falle zeichnet hauptsächlich die Weiterentwicklungen des leistungsstarken Mikrochips und des Computers dafür verantwortlich. Die Abtastung analoger elektrischer Signale und deren Umwandlung und Speicherung als diskrete digitale Werte führen zu einer qualitativen Überlegenheit und weitaus höheren Flexibilität der Digitaltechnik gegenüber der Analogtechnik, die zuvor nur die direkte Aufzeichnung von Audiosignalen auf Rekordern erlaubt und fachmännisch nur in professionellen Musikstudios unter erheblichem finanziellen Aufwand möglich war. Allen voran die sogenannten Musikcomputer in Form von digitalen Synthesizern, einer Weiterentwicklung ihrer analogen Pendants, die bereits seit den 60er Jahren von Pionieren wie Bob Moog entwickelt wurden und mittels Oszillatoren Klangerzeugung ermöglichten, sind hier zu nennen. Zu den neuen Talenten dieser technischen Entwicklung, die in etwa Anfang 1980 beginnt, zählen unter anderem auch die Sampling- und Sequencing-Fähigkeiten neuer musiktechnischer Workstations, die zunächst digital Töne berechnen um sie danach als analoge Signale über Boxen und Kopfhörer hörbar auszugeben. (vgl. HDM STUTTGART 2008)

Zur klassischen Begabung der digitalen Synthesizer zählt die Produktion völlig neuer Töne und Geräusche via Klangsynthese in verschiedenster Form, auf die hier in diesem Rahmen nicht näher eingegangen werden kann. Es sei die bekannteste dieser Varianten genannt, die Frequenzmodulation, bei der sich zwei oder mehrere Oszillatoren verschiedener Sinusschwingungen in Abhängigkeit von einem bestimmten Algorithmus gegenseitig modulieren und dadurch komplexe und neuartige Klangfolgen entstehen. Abbildung 3.1 zeigt den

in den 80er Jahren weit verbreiteten DX 7 Synthesizer der Firma Yamaha, der in nahezu allen Popmusikproduktion dieser Zeit zu hören ist. (vgl. SEQUENZER.DE 2011)



Abb. 3.1: Yamaha DX-7 Synthesizer

Sequencing hingegen beschreibt ursprünglich die Fähigkeit mittels Steuerdaten verschiedene Klangerzeuger wie beispielsweise Synthesizer anzusteuern. Diese Daten bestehen unter anderem aus der zeitlichen Abspielposition, der Dauer oder der Lautstärke einzelner Töne (vgl. GÖRNE 2008). Um eine Vereinheitlichung dieser Steuerdaten zu garantieren wurde das MIDI Format eingeführt, das in Kapitel 4.2.1 sehr ausführlich behandelt wird. Die aber wohl weitreichendste Neuerung beschreibt die Sampling Technologie. Hierbei wird Audio-Material aufgenommen und gespeichert, um danach, durch verschiedene Bearbeitungsformen verändert und verfremdet oder einfach ohne Manipulation, beliebig oft verlustfrei wiedergegeben zu werden. Es ist demnach also auch möglich verschiedene mehr oder weniger kurze Teile älterer musikalischer Stücke zu „samplen" und diese in eigene Produktionen einfließen zu lassen. In Abbildung 3.2 ist der MPC 60 zu sehen, das Urgestein der AKAI Sampler Familie, die zum festen Bestandteil in vielen Studios geworden ist. Durch das Sampling wurden viele verschiedene Musikstile entscheidend geprägt, vor allem bei den Genres Rap und Hip-Hop wird damit sehr oft gearbeitet. Ein gutes Beispiel bietet das Lied



Abb. 3.2: Akai MPC 60 Sampler

„Funky Drummer" von James Brown, das nach THE-BREAKS.COM in 184 anderen Musikproduktionen verwendet wird und damit wohl den Rekord als meist-gesampletes Lied innehält. (vgl. SMUDITS 2008)

Die vorgestellten neuen Instrumente wie Synthesizer, Sampler und Sequenzer beschreiben nur die grundlegenden Neuerungen, die mit der Digitalisierung eingeführt wurden. Zusätzlich wurden digitale Effektgeräte entwickelt, um zum Beispiel durch Hall oder Echo Räumlichkeit simulieren zu können, mittels Kompressoren die Dynamik zu bearbeiten oder mit Hilfe von Filtern und Equalizern das Frequenzspektrum zu manipulieren. Diese Neuentstehungen gipfelten letztendlich in zahlreiche Mischformen dieser Techniken wie Workstations oder Drumcomputer, so auch die TR Serien von Roland (siehe Abbildungen 3.3 und 3.4), die neue Musikstile wie House, Dance, Rap, Synthie-Pop, Techno und etliche mehr bis weit in die 90er Jahre hinein prägten (vgl. STANGE-ELBE 2008). SMUDITS (2008) behauptet im Hinblick auf die soeben kurz zusammengefasste, in seinen Worten genannte, digitale Mediamorphose sogar, „dass sich diese Transformaton noch zu Beginn des 21. Jahrhunderts in ihrer Anfangsphase befindet".



Abb. 3.3: Roland TR-808



Abb. 3.4: Roland TR-909

3.1.2. Virtualisierung

Die Virtualisierung kann durchaus als logische Folge der Digitalisierung angesehen werden. So definiert STANGE-ELBE (2008) das

> „Virtuelle als die Eigenschaft eines Sachverhaltes, der nicht in der Form existiert, in der er zu wirken scheint, aber in seinem Wesen und seiner Wirkung eines real existierenden Sachverhaltes gleichartig ist [...und demnach...] lassen sich digitalisierte Klänge bereits als virtuelle Klänge definieren."

Die eigentliche Neuheit besteht nun darin, dass die komplette Studiotechnologie auf die virtuelle Ebene des PCs transformiert wird. Alle Klänge, Synthesizer, Sampler, Mischpulte wie auch Effektgeräte werden dabei komplett im Computer berechnet und auf seine grafische Benutzeroberfläche (GUI) auf angeschlossenen Monitoren projiziert. Viele Instrumente und Klangmodulatoren wie auch deren reale Oberfläche sind damit vollständig in Form von sogenannten virtuellen Instrumenten vorhanden. Dabei können das beispielsweise wie erwähnt Nachbildungen von analogen oder digitalen Synthesizern sein, komplett neu erschaffene reine Softwaresynthesizer ohne reale Entsprechung oder auch Software-Analogons jeglicher klassischer Musikinstrumente, sogar komplette Sinfonieorchester. Darüber hinaus bieten diverse Programme auch die Möglichkeit in der virtuellen Umgebung eigene Instrumente zu entwickeln. Reaktor von Native Instruments beispielsweise erlaubt wie im unteren Teil der Abbildung 3.5 sehr gut zu sehen ist, die Konzeption eigener Synthesizer durch Kombination und Verschaltung verschiedener Bausteine die Oszillatoren, Effekte, Schaltungen und vieles mehr repräsentieren und damit zu fast unendlich vielen Möglichkeiten verschiedener Klänge führen. Abbildung 3.6 zeigt weiterhin den Arturia Moog Modulator, ein Software Nachbau des im vorigen Kapitel angesprochenen analogen Ur-Synthesizers von Robert Moog. (vgl. HDM Stuttgart 2008)

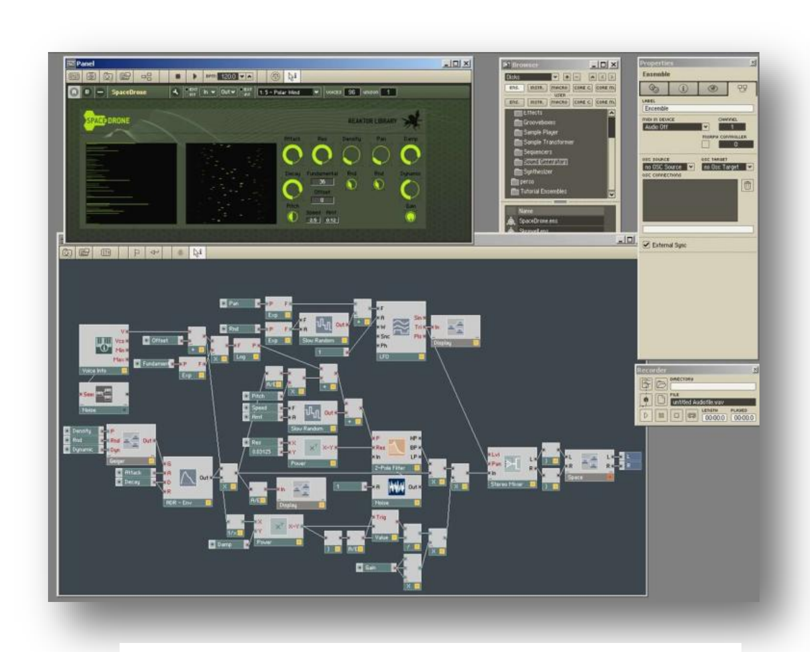


Abb. 3.5: Native Instruments Reaktor



Abb. 3.6: Arturia Moog Modulator

Der allerdings noch wichtigere Schritt zur Musikproduktion in virtueller Umgebung ist mit der Einführung von speziellen Digital Audio Workstations, kurz DAW, erfolgt. Diese DAWs sind Software-Nachbildungen von großen Teilen eines kompletten Musikstudios. Sie stellen letztendlich eine virtuelle Studio-Umgebung inklusive Mischpult, Arrangierfenster und verschiedenen anderen Editoren dar, die im Zusammenspiel eine professionelle Musikproduktion ermöglichen und damit teure Studio-Mischpulte und Aufnahmegeräte ersetzen. Darüber hinaus enthalten sie aber auch noch viele der soeben beschriebenen virtuellen Instrumente und überdies auch Effekte wie zum Beispiel Equalizer, Hall oder Kompressoren. Dabei kann nicht nur DAW-

eigene Software verwendet werden. Auch sogenannte Plug-In-Effekte und -Instrumente von anderen Anbietern können über festgelegte Schnittstellen wie Audio Unit (AU) oder Virtual Studio Technology (VST), die alle mittels des MIDI Protokolls kommunizieren und angesteuert werden, eingebunden werden. Weiterhin ist natürlich auch die Bearbeitung von Audio Material möglich. Eine der bekannteste aber bei weitem nicht die einzige DAW, Logic Studio 9, zeigt Abbildung 3.7. Hier ist im oberen Bereich das Arrangierfenster mit den einzelnen Spuren zu sehen und im unteren Teil das zugehörige Mischpult. (vgl. MAIER 2008)



Abb. 3.7: Digital Audio Workstation Logic Studio 9

Zusammenfassend lässt sich hier bemerken, dass der Personal Computer in den Mittelpunkt des Geschehens der Musikproduktion rückt und als universelles Produktionsmittel fungiert. Er vereint in virtueller Form, bis auf die Tonausgabe für die eventuell gesonderte Lautsprecher angeschlossen werden müssen, alle notwendigen Funktionen durch verschiedenste Arten von Programmen.

3.1.3. Auswirkungen von Digitalisierung und Virtualisierung

Aus der soeben benannten Tatsache heraus folgen etliche Vor- aber auch Nachteile, die im Folgenden kurz erläutert werden. Dieser Überblick legt hierbei jedoch keinen Anspruch auf Vollständigkeit und Ausführlichkeit, da dies im Rahmen dieser Arbeit unangebracht wäre. Es werden vielmehr nur die wichtigsten dieser bewertenden Elemente aufgezählt.

Der wohl meistgenannte und wichtigste Vorteil ergibt sich hinsichtlich des finanziellen Standpunktes. Der Kaufpreis ist mit dem realen Studioequipments nicht zu vergleichen und alle Komponenten können problemlos auch in einem sogenannten Home-Studio aufgebaut und miteinander kombiniert werden. Weiterhin ergibt sich eine wesentliche Steigerung der Flexibilität. Klangparameter lassen sich in DAW-Programmen unbegrenzt speichern, automatisieren und bearbeiten. Eingaben und Bearbeitungsschritte können beliebig oft wiederholt sowie widerrufen werden. So gut wie alle Instrumente sind zu jeder Zeit direkt verfügbar. Zudem sind auch nicht mehr vorhandene originale Instrumente mit Hilfe von virtuellen Nachbildungen wieder zugänglich. Inwieweit die Klangqualität sich mit den jeweiligen Originalen allerdings vergleichen lässt ist fraglich. Unbestritten ist jedoch, dass die virtuellen Instrumente, im Gegensatz zu ihren realen Entsprechungen, ohne große Expertise werden können und somit auch für den musikalischen Laien zugänglich sind. (vgl. ClAUSSEN 2006)

Dabei drängt sich jedoch vor allem die Frage nach sinnvoller Bedienung dieser Software-Instrumente auf. Grundsätzlich ist die Bearbeitung und Erstellung von MIDI Daten per Maus und Tastatur in diversen Editoren möglich. Abbildung 3.8 zeigt einen dieser Editoren in Logic, die Piano Roll, in der die einzelnen Noten in Form von kleinen Bausteinen angezeigt werden. Die Farbe der MIDI Events wie die Noten in MIDI Form auch genannt werden, spiegelt die Velocity wieder, die die Anschlagsdynamik einer einzelnen Note repräsentiert.

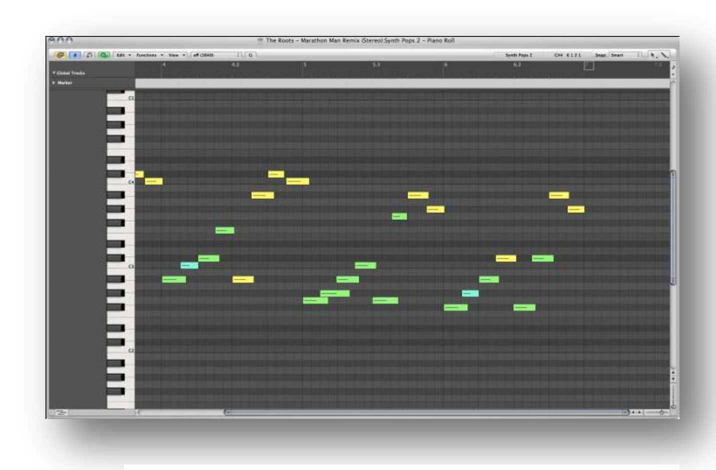


Abb. 3.8: Piano Roll in Logic Studio

Da diese Form der Eingabe jedoch weit entfernt ist von der ursprünglichen und natürlichen Form, die instrumentales Spiel und prinzipiell auch Musik an sich auszeichnet und darüber hinaus den Definitionen von Intuitivität und Spontanität widerspricht, die oftmals dem Ursprung einer musikalischen Komposition

entsprechen, sind neue Formen der Eingabe für Software-Instrumente entwickelt worden, die sogenannten MIDI-Controller (vgl. Chadabe 1997). Da diese von grundlegender Bedeutung in dieser Arbeit sind, widmet sich Kapitel 3.2 eigens diesen Steuerungseinheiten.

Bei all den beschriebenen Funktionen, die die vorgestellten Programme beherrschen müssen um als Ersatz für professionelles Studioequipment zu dienen, ergibt sich ein zweiter großer Nachteil vor allem im Umgang mit den Digital Audio Workstations: die ungeheure Komplexität und Unübersichtlichkeit der Software und ihre daraus resultierende manchmal schwierige Bedienung (vgl. MEIER 2008). Allein der Umfang der Handbücher, die nicht selten mehr als tausend Seiten beinhalten, spricht hier für sich und obwohl an der Erleichterung der Bedienung gearbeitet wird und ebenfalls Video-Tutorien dazu angeboten werden, kann sich das Erlernen solcher Software als zeitintensives Unterfangen gestalten. Inwieweit dadurch der eigentlich künstlerische Aspekt des Musizierens verloren geht und dabei von der reinen Kompetenz zur Bedienung von Technik und Software verdrängt wird, versuchen die folgenden Kapitel zu klären.

3.1.4. Gesellschaftliche und musikalische Auswirkungen

Um nunmehr einen vollständigen Blick auf das Thema der digitalen Musikproduktion werfen zu können, fehlt neben der in den vorigen Kapiteln zusammengefassten geschichtlichen Entwicklung, der Beschreibung der Möglichkeiten und deren Vor- und Nachteile nun noch ein Wort zu den Auswirkungen, die aus dem Digitalisierungs- und Visualisierungsprozess resultieren. Diese sind zum Teil eng miteinander verbunden bzw. bedingen sich gegenseitig.

3.1.4.1. Verlust von Referenzierbarkeit des Klangs

Aus musiktheoretischer Sicht stellt der Verlust einer spezifischen Referenz des Klangereignisses, welche nun erläutert wird, die wohl erheblichste Konsequenz dar. Die Ursache von Schall, in Form von Geräusch, Klang, Ton oder Knall, ist grundsätzlich immer eine Schwingung eines Gegenstandes. Die darauf folgende wellenförmige Ausbreitung von Druck- und Dichteschwankungen in einem elastischen Medium wie Luft wird dann eben als Schallwelle bezeichnet und kann vom Menschen wahrgenommen werden. Dabei hängt die Charakteristik eines Klangs von drei verschiedenen Umständen ab: der Klangquelle, dem Rezipienten und der Räumlichkeit (vgl. CLAUSSEN 2006 und GÖRNE 2008). Wird nun das subjektive Hörempfinden des Einzelnen ausgeschlossen, so definiert sich die Einzigartigkeit eines Klangereignisses über seine Quelle und den Raum, in dem es produziert wird.

Wie verändert sich dabei aber dieses Prinzip auf der digitalen und virtuellen Ebene? Dazu gibt ein Blick auf die neuen technischen Möglichkeiten Aufschluss. Mit den beschriebenen Hardware-Synthesizern wie auch Software-Instrumenten lässt sich nahezu jeder Klang simulieren und imitieren. Der Ursprung eines jeglichen Tons oder Geräuschs liegt dann in von Schaltkreisen implizierten Spannungsschwankungen und wird durch Lautsprechermembranen erzeugt. Eine direkte reale Entsprechung des erzeugten Sounds gibt es nicht mehr. Insbesondere instrumentale Eigenschaften wie die Klangfarbe oder der Ausdruck des Instrumentalisten bei

seinem Spiel, so auch die Virtuosität, wird durch Parameter der MIDI-Steuerungsdaten ersetzt. (vgl. SMUDITS 2008) Nach STANGE-ELBE 2008 wird hierbei „das Prinzip von Ursache und Wirkung durchbrochen“ und die eigentliche Referenz geht verloren. Vielmals werden so auch mit den Standard-Peripheriegeräten Maus und Tastatur oder einem angeschlossenen Keyboard alle Formen der meist Sample-basierten virtuellen Instrumente gespielt, womit der eigentlich künstlerische Aspekt verkommt und der Bezug zu einem Instrument nicht mehr hergestellt werden kann. Ganz zu Anfang dieser Evolution steht die Erfindung der Sampling-Technologie, die gewissermaßen die Verschmelzung von Instrument und Wiedergabegerät verkörpert und mit der es möglich wurde Klänge aller Art, oftmals auch Teile älterer Lieder, aufzuzeichnen, zu bearbeiten, neu zusammenzusetzen und in eigene Produktionen einfließen zu lassen (siehe Kapitel 3.1.1). Die eindeutige Zuweisung eines Samples zu einem gewissen Musikstück ist damit nicht mehr möglich und somit wird auch ein Referenzieren dazu ausgeschlossen. (vgl. GROSSMANN 1995)

Aber nicht nur die Quelle des Klangs ist damit nicht mehr zweifelsfrei nachvollziehbar, auch die zweite Charakteristik eines Klangs, der Raum, geht im virtuellen und digitalen Umfeld verloren, denn mit Hilfe von Effekten wie beispielsweise Hall oder Delay, lassen sich auf jederlei Sound auch alle erdenkbaren Räumlichkeiten projizieren. Diese können einem wirklichen Raum nachempfunden sein, aber auch surreale Formen annehmen: „Die Konstruktion fiktiver Räume und musikalischer Binnenarchitekturen war angesichts dieser referenzlosen elektronischen Sounds nur mehr unter Hinzuziehung kosmischer oder psychedelischer Raumkonzepte möglich“ (SMUDITS 2008). Ein Beispiel dafür ist auch das sogenannte Physical Modeling, mit dem sich prinzipiell alle möglichen, auch in der Realität nicht umsetzbaren Ausformungen von Klangkörpern in jeglichen Raumkonzepten simulieren lassen und keine physikalischen Einschränkungen mehr gegeben sind. Somit entstehen „abstrakte hybride Instrumentenmodelle, die nur noch mit Metaphern, wie gestrichene Flöte (die Luftsäule eines Blasmodells wird mit einem Bogen erregt) […] oder geblasenes Cello (das Modell einer Seite, die durch anblasen in Schwingung versetzt wird) [usw.]“ (HARENBERG 2003) beschrieben werden können.

Es lassen sich zu Abschluss dieses Kapitels demnach zwei Entwicklungen erkennen. Die Erste besteht darin, dass in der digitalen und virtuellen Musikproduktion zwar oftmals analoge Instrumente ihrem realen Vorbild nach möglichst ebenbürtig simuliert werden, diese aber nicht mit den üblichen Eingabegeräten eines Computers angebracht gespielt werden können und dadurch die Kreativität und Intuitivität, die musizieren und vor allem den Kompositionsprozess identifiziert, verloren geht. Gleichzeitig jedoch werden ebenfalls immer neuartige Sounds gesucht und erschaffen, die, wie oben beschrieben, keine oder kaum eine Entsprechung in der Realität vorweisen können, aber dennoch immer stärker in neuartigere Musikformen mit einfließen. Eine Möglichkeit, den angesprochenen Tendenzen der Tonerzeugung, die die digitales Musikproduktion nach sich zieht, gerecht zu werden, wäre es, neue entsprechende Eingabemedien zu entwickeln, die den kreativen Ideen wieder Zugang ermöglichen und die angesprochene Virtuosität auch mit virtuellen Instrumenten, egal ob Nachbildung oder Neuerfindung, gestatten.

3.1.4.2. Demokratisierung und De-Professionalisierung der Musikproduktion

Zusätzlich zu der veränderten Bedingungen aus musikalischer Sicht heraus, verändern sich auch einige eher gesellschaftliche Aspekte, die mit der Produktion von Musik verbunden sind.

Etwa ab Anfang der 1990er Jahren ist „festzuhalten, dass eine ständige Miniaturisierung und Verbilligung der Produktionsmittel von Musik zu beobachten ist, sodass es [...] möglich wird, ohne allzu großen Kostenaufwand eine den professionellen Standards entsprechende Musikproduktion gleichsam im eigenen Haushalt herzustellen“ (SMUDITS 2008). Hinzu kommt, dass der Musikschaffende wie bereits erwähnt mit den Mitteln der Produktionsmethoden im Virtuellen „weder auf die Imagination seines Werkes im Vorhinein und seine Transkription, noch auf die Beherrschung traditioneller Instrumente angewiesen“ (CLAUSSEN 2006) ist. Durch die Distribution von Musik über das Internet und der Abkehr vom materiellen Tonträger ist fast ein jeder in der Lage seine Werke selbstständig und nahezu risikofrei weltweit zu vermarkten. Das Resultat dieser Entwicklung, bedingt durch die soeben aufgezeigten Faktoren, besteht darin, dass es nun für jedermann möglich ist unter praktisch professionellen Bedingungen eigene Musik zu kreieren. Bei dieser Entwicklung kann dann von einer Demokratisierung musikalischer Produktionsbedingungen gesprochen werden. (vgl. ENGH 2008)

Darüber hinaus findet ein zweiter Trend statt, der auch als De-Professionalisierung bezeichnet wird. Wie auch im vorigen Kapitel schon angesprochen, sind mit Hilfe der Digital Audio Workstations und der virtuellen Instrumente keine traditionellen musikalischen Kompetenzen mehr notwendig. Nach GROSSMANN (1995) leistet der Computer „heute eine so weitgehende Integration von Vorwissen sowie komplexer und veränderbarer Vorstrukturierung elektronischer Klangproduktion, dass auch vom musikalisch-handwerklich unausgebildeten Laien komplexe Klänge gespielt werden können.“ Daraus leitet SMUDITS (2008) ab, dass in „Bezug auf das Berufsbild von Musikschaffenden [...] eine Aufweichung der professionellen Zugangskriterien in das künstlerische Feld konstatierbar“ ist.

Auch das Verhältnis von Rezeption und Produktion des Einzelnen verändert sich prinzipiell, denn mit den neuen Technologien sind Bedingungen geschaffen, die eben den Einstieg in das sogenannte Musikbusiness stark erleichtern. Dabei ist beispielsweise oftmals die Tätigkeit als Discjockey der erste Schritt. DJ-Programme wie Traktor Scratch von Native Instruments, einer Software die als kompletter digitaler Ersatz der Schallplattenspieler fungiert, erlaubt das Einbinden jeglicher Musikdateien auf dem PC und bereits dadurch eine Art musikalisch kreative Handlung (vgl. STANGE-ELBE 2008). Abbildung 3.9 zeigt verschiedene Stufen und die Verbindung zwischen Konsum von Musik und dem eigentlichen Musizieren, die sich durch die genannten Entwicklungen stark verändert hat.

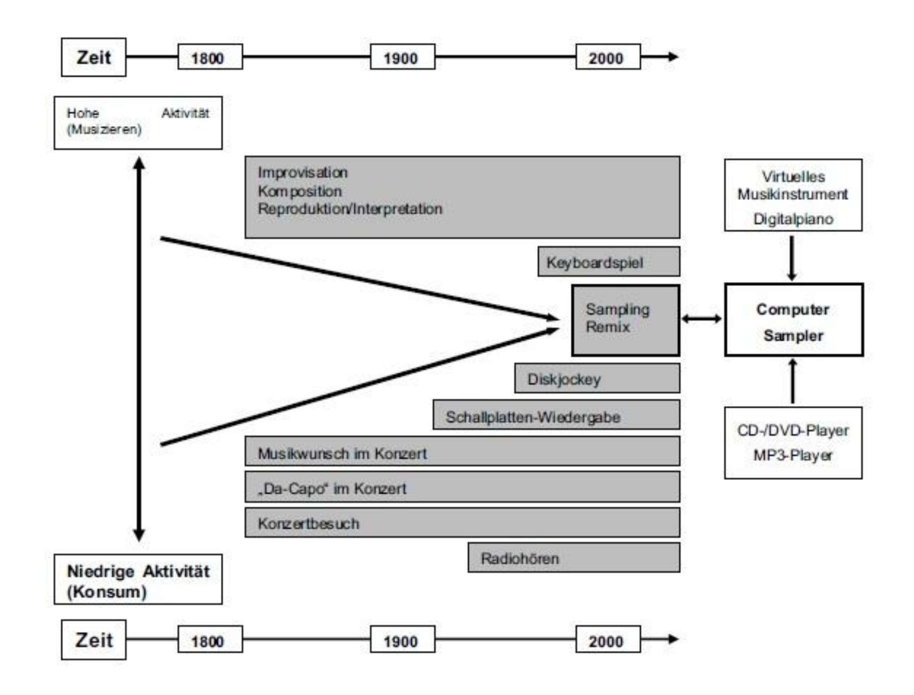


Abb. 3.9: Musikkonsum und Musizieren

All die dargelegten Tendenzen verändern somit auch das klassische Bild des Künstlers im Musikbereich. Denn durch „die Möglichkeit der klanglichen Simulation in Echtzeit wird das digitale Studio zum Kompositionswerkzeug, Komponist, Musiker und Toningenieur zum Produzenten. Die traditionelle Trennung zwischen Produktion, Realisation und Rezeption wird aufgehoben" (CLAUSSEN 2006). Der Computer und die DAWs rücken in den Mittelpunkt des Geschehens und es entsteht ein neuer universeller Typus des Musikschaffenden, der neben der reinen Produktion auch Interpretation, Komposition und wie oben kurz beschrieben sogar die Distribution von Musik in sich vereinen kann und somit vom Instrumentalisten unabhängig wird.

> „Zusammenfassen lässt sich diese Tendenz unter dem Schlagwort der flexiblen Spezialisierung, bei der die Musikschaffenden frei von unmittelbaren musikindustriellen Vorgaben werden, aber dafür umso mehr auf ihre Selbstbehauptung am freien Markt angewiesen sind. Der qualitative Wandel geht also von der Musikproduktion unter Bedingungen der Kulturindustrien zu neuen, weniger künstlerisch, als vielmehr unternehmerisch autonomen Formen von Musikproduktion" (SMUDITS 2008).

Eine weitere Folge der beschriebenen Demokratisierung besteht außerdem darin, dass sich Firmen aufgrund steigender Nachfrage mit ihren Produkten nun verstärkt nach den Bedürfnissen des Universal-Talents Produzent als neue Zielgruppe richten (vgl. GROSSMANN 1995). Wie schon im vorigen Kapitel erwähnt benötigen die neuen Formen der Instrumente passende Eingabeinterfaces, die gebrauchstaugliche Steuerung ermöglichen aber auch eine künstlerische Verbindung herstellen. Das Folgende Kapitel widmet sich darum eingehend diesen sogenannten Controllern.

3.2. Neue musikalische Interfaces

In den letzten beiden Kapiteln ist resultierend aus den Veränderungen der Musikproduktion dargelegt worden, dass der physische Zugang beim Arbeiten mittels Simulationen bezüglich Tonerzeugung allmählich verloren geht. Diese Distanzierung vom körperlich gestischen Zugang verdrängt wie bereits erwähnt eben auch die den künstlerischen Ausdruck und das individuelle Spiel eines Instruments. Vielmals wird auf virtueller Basis komponierte Musik darum als mechanisch und steril empfunden. Deshalb geht es nunmehr um „die Konzeption fortgeschrittener Mensch-Medium Schnittstellen oder Interfaces, die einen Austausch menschlicher Befindlichkeit mit den inneren und äußeren Zuständen elektronischer Medien erlauben. Sie sollten zu einer adäquaten Nutzung medialer Möglichkeiten durch neue Instrumente bzw. Spielfelder und zu einem neuen Verhältnis von Produktion und Rezeption führen" (GROSSMANN 1995), wobei aber gleichzeitig berücksichtigt werden muss, dass durch den erläuterten Demokratisierungsprozess die Eingabemöglichkeiten auch dem musikalisch nicht ausgebildeten Anwender als intuitives Medium dienen. STANGE-ELBE (2008) sieht hierin „die Zukunft einer neuen Instrumententechnologie", wobei diese nach GROSSMANN (1995) wie auch das traditionelle Instrumentarium „die Aufgabe [haben], möglichst nahe an die Motorik und Zeitstruktur einer menschlichen Aktion heranzukommen" Aus den obigen Darlegungen heraus erscheint es also sinnvoll, ein musikalisches Interface auf der einen Seite als eine Art Werkzeug benutzen zu können, mit dem die Steuerung der komplexen Strukturen und Abläufe in den musiktechnischen Softwarelösungen möglich wird. Auf der anderen Seite sollte die Schnittstelle aber auch als ein Medium fungieren, das gleichzeitig den künstlerisch-virtuosen Spielraum bereitstellt, der den direkten Bezug und die beschriebene Körperlichkeit sowie den Ausdruck des Musikspiels erlaubt.

Um von dieser Betrachtung ein Bild zu bekommen, werden im Folgenden verschiedene Controller beispielhaft vorgestellt, ohne dabei auf ihre genauen technischen Spezifikationen einzugehen. Zum Teil handelt es sich um fertige und vertriebene Produkte, es werden aber auch Studien auf eher experimenteller Basis gezeigt. Da mittlerweile der Markt dieser Interfaces enorme Ausmaße angenommen hat, kann hier kein vollständiger Überblick gegeben werden. Vielmehr werden nur ausgewählte, wichtige und interessante Beispiele gezeigt.

Zunächst sind hier die standardisierten und universellen Interfaces zu nennen. Sie bieten eine taktile Steuerung verschiedener Software und gelten daher als recht flexibel anwendbar. Fader oder sogenannte Pads (siehe Abbildungen 3.10 und 3.12) lassen sich per MIDI-Befehl einzelnen oder mehreren Parameter zuordnen und erlauben dadurch ihre Hardware-Bedienung. Zu dieser Gruppe zählen auch die MIDI-Keyboards, abgebildet in Abbildung 3.11, die sehr häufig verwendet werden um alle Arten virtueller Instrumente mit Hilfe einer Klaviatur zu spielen. Alle Eingabegeräte dieser Art haben jedoch eher Werkzeug Charakter, der zwar die Kontrolle beschriebener virtueller Audiotechnik sehr universell unterstützt und gewährleistet, die musikalische Ausdrucksweise aber noch stark limitieren. (vgl. ClAUSSEN 2006)



Abb. 3.12: Pad-Controller   Abb. 3.11: MIDI-Keyboard   Abb. 3.10: Fader-Controller

Eine neuere Spezies dieser Schnittstellen sind die Touchpad-Controller, die insbesondere mit den Smartphones und Tablet-PCs immer interessanter werden (vgl. WEBER 2010). Hiermit können die Regler, die ebenfalls Software-Parametersteuerung erlauben über den berührungsempfindlichen Touchscreen des Produkts bedient werden. Ein großer Vorteil ist die Individualisierbarkeit der Bedienoberfläche, sodass Fader, Dreh-Regler, Pads, eine Klaviatur oder Sequencer-Steuerungselemente nach eigenen Vorstellungen in verschiedenen Zusammenstellungen in einem eigenen Editor kombiniert werden können. Zumeist werden die Controller-Programme als sogenannte Apps beispielsweise für iPhone und iPad angeboten und liefern, ohne den Kauf eines Touchscreen-Produkts zu berücksichtigen, für relativ wenig finanziellen Aufwand eine sinnvolle und auch flexible Lösung. Eines dieser Programme für Apple Produkte ist TouchOSC der Firma Hexler. (vgl. HEXLER.NET 2011) In einem Screenshot in Abbildung 3.13 ist eine Kombination verschiedener Steuerungselemente zu sehen. Zusätzlich sei auf das Video der Seite PIXIL.INFO (2011) verwiesen, auf dem mittels iPad und TouchOSC die DAW Logic und diverse Software-Instrumente bedient werden.



Abb. 3.13: TouchOSC Screenshot

Wiederum eine ganz andere und kreative Schnittstellen-Lösung zeigt die in Abbildung 3.14 dargestellte „Ziggybox“, ein Projekt der Digitales-Mediendesign Studenten Christian Losert und Paul Schengber an der Hochschule Darmstadt, mit der durch Bewegung von Zigaretten und deren Schachteln, die mit Hilfe von Lichtsensoren erkannt werden, die Software Ableton gesteuert und moduliert wird. (vgl. DE-BUG.DE 2011)



Abb. 3.14: Ziggybox MIDI-Controller

Als letztes Beispiel zu Controllern verschiedener Art dient der „Reactable“, der in Abbildung 3.15 zu sehen ist und erstmals bei einem Live-Auftritt der Künstlerin Björk im Jahr 2007 zu bestaunen war. Dieses von der Music Technology Group der Universität Pompeu Fabra entwickelte System besteht aus einer lichtdurchlässigen Multitouch-Tischoberfläche, die von unten das Bewegen verschiedener kleiner Blöcke und der Finger durch Kameras analysiert um dadurch die Klanggestaltung in Echtzeit zu steuern. (vgl. REACTABLE.COM 2011) Die Blöcke repräsentieren dabei Komponenten eines modularen Synthesizers, den der User gleichzeitig spielt und zusammenbaut, indem er während der Wiedergabe Objekte auswählt und in Relation zueinander auf der Oberfläche positioniert und bewegt. Spielen und Erstellen des Instruments sind im Vergleich zur Reaktor-Software (siehe Kapitel 3.1.2) keine getrennten Vorgänge mehr. Gleichzeitig gibt der Reactable visuelles Feedback, das infolge der Aktion erzeugt wird. Trotz der insgesamt komplexen Technik ist die Handhabung intuitiv, spielerisch und leicht erlernbar. Physische Objekte, Visualisierung, Klang und Aktion des Anwenders stehen in unmittelbarer Relation zueinander. (vgl. JORDA 2005)

Im Gegensatz zu dem in diesem Kapitel zuerst vorgestellten Standard-Controllern, soll an diesem Beispiel deutlich werden, wie der Zugang zu simulationsbedingter Musikerzeugung erweitert werden kann. Unter

REACTABLE.COM TUTORIALS (2011) finden sich diverse sehr eindrucksvolle, den Reactable erklärende Videos. Einen Nachteil stellt womöglich der Preis dieses Interaktionsgeräts mit rund 10000 Dollar dar. Doch dafür wird mittlerweile eine Reactable-Mobile App angeboten, die alle Funktionen des Reactables auch auf dem iPad oder iPhone Touchscreen ausführt.



Abb. 3.15: Der Reactable

Die Bandbreite wie auch die Unterschiede der verschiedenen Eingabegeräte sind ziemlich beträchtlich. Der Markt hierfür wächst aufgrund der steigenden Nachfrage durch den anhaltenden Prozess der Demokratisierung kontinuierlich. Allerdings werden vor allem genau die in den ersten Beispielen gezeigten Interfaces, die eher auf Universalität ausgerichtet sind und mehr als Werkzeuge für die digitale Ebene dienen, noch am häufigsten vertrieben. Dass indessen aber auch durchaus Bedienelemente konzipiert werden, die Spielräume für Kreativität und Intuition lässt, sollten die anderen Beispiele zeigen.

4. Technische Grundlagen

Bevor ein passendes Konzept entworfen werden kann, ist es nahezu unabdingbar, zunächst festzulegen, aus welchen verschiedenen technischen Bestandteilen sich das Gesamtsystem zusammensetzt. Dazu zählt die Wiimote-Fernbedienung, die zwei Interfaceprotokolle und Datentransferverbindungen MIDI und Bluetooth und zuletzt noch die Software, die als Klangerzeuger letztlich verwendet wird.

4.1. Die Wiimote

Schon früh während des Prozesses der Ideenfindung ist die Entscheidung über das Aufnahmegerät der Eingabebewegung gegen andere Interface-Einheiten wie Touchpad oder Maus zugunsten des Nintendo Wii Controllers gefallen, da die sogenannte Wii Remote in diesem Fall eine ganze Reihe von Vorteilen bietet, die in diesem Kapitel erläutert werden. Im ersten Moment erscheint das Eingabemodul einer Spielekonsole vielleicht ungeeignet für ein System zur intuitiven Musikproduktion, doch Kontrolleinheiten von Videospielkonsolen spielen eine immer größere Rolle beim Design von gerade durch junge Nutzer häufig verwendeten Produkten, da diese von klein auf wortwörtlich spielerisch mit dem Umgang sogenannter Gamepads aufwachsen und darum häufig verwendete Gesten, Symbole oder Kombinationen und Abläufe, die beim Videospiel notwendig sind, von selbst erkennen und verwenden können.

Prinzipiell besteht der Wii Controller aus zwei Teilen: Der Hauptteil der Steuerungseinheit, im Folgenden nur noch als Wiimote bezeichnet, und die daran anschließbare sogenannte Nunchuk-Erweiterung (siehe Abbildung 4.1).



Abb. 4.1: Wiimote und Nunchuck

Neben den üblichen Eingabemöglichkeiten über Knöpfen und Analogsticks, verfügt die Wiimote ebenfalls über eine völlig neue Art der Eingabe über zwei unterschiedliche Sensortechniken. Wer in seiner Freizeit bereits einmal die Wii-Spielekonsole benutzt hat, stellt fest, dass bei vielen der Videospiele die Steuerung durch Schütteln des Controllers oder einer ähnlichen Schlagbewegung erfolgt wie beispielsweise bei Nintendo

Wii Sports Resort in der Disziplin Golf. Zudem kann bereits die Haltung des Controllers Auswirkungen auf manches Videospiel zeigen und somit als Eingabe dienen. Dafür sind die linearen Beschleunigungssensoren der beiden Kontrolleinheiten zuständig, die die jeweils auf die Wiimote und den Nunchuck wirkende Beschleunigung auf drei Achsen eines in die Fernbedienungen projizierten, festen Koordinatensystems transformiert, das in Abbildung 4.2 verdeutlicht ist. Durch diese Transformation der Beschleunigung kann die Lage der ruhenden Wii Remote genau bestimmt werden, da sich somit die Erdbeschleunigung in die verschiedenen Achsenrichtungen aufteilt und sich daraus wiederum die Winkel berechnen, die eine Lage im Raum beschreiben (vgl. WIIBREW.ORG). Zeigt zum Beispiel die komplette Beschleunigung in negative Z-Richtung, so liegt die Wiimote mit der Knopfseite nach oben. Weiterhin kann dieser Sensor natürlich auch jede andere Form von Bewegungen mit einer Genauigkeit von 0,3g im Spektrum von -3g bis +3g, die mit dem Wii Controller ausgeführt werden, erfassen und verarbeiten.

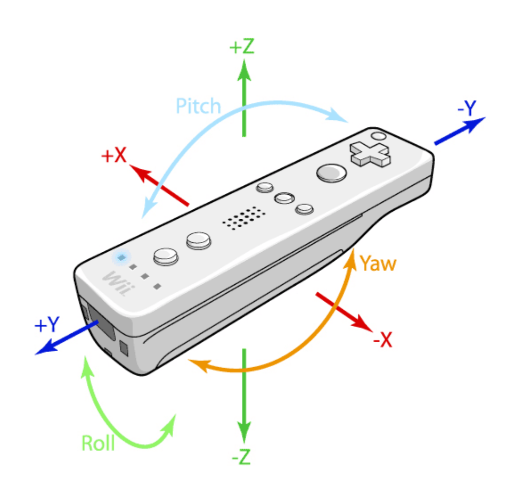


Abb. 4.2: Wiimote Bezugssystem

Die Wiimote verfügt zusätzlich noch über einen Positionssensor. Dieser erfasst mit Hilfe einer Infrarotkamera, die an der Spitze der Wiimote installiert ist, bis zu vier Infrarotquellen und kann somit die relative Lage bestimmen. Natürlich kann die soeben beschriebene Kamera bei einer ruhenden Wiimote auch verwendet werden, um Positionen, Geschwindigkeiten und Beschleunigungen von sich beispielsweise im Raum bewegenden Infrarotdioden zu detektieren. Auf diese Möglichkeit der Signalerfassung und -verarbeitung wird das zu entwickelnde System letztlich aufbauen und daher wird zu einem späteren Zeitpunkt noch einmal genauer darauf eingegangen werden.

Neben den Beschleunigungssensoren bietet die Wiimote als Sensor zudem noch eine Infrarotkamera an ihrer Spitze, die mit einer Auflösung von 1024x768 Bildpunkten vier Infrarot (IR) Signale gleichzeitig erfassen kann (vgl. MULB.ORG 2011). Damit ist Wii-Remote in der Lage, ihre Position in Bezug auf diese Infrarotpunkte zu berechnen. Normalerweise wird die Sensorik an der Spielekonsole benutzt, um mittels der Fernbedienung eine, der Maus auf dem PC ähnliche, Zeiger-Funktion auf dem Bildschirm zu gewährleisten um damit zu navigieren und auszuwählen. Falls die Wiimote in einer Ruhelage platziert wird, kann diese Sensorfunktion auch zur Detektion von im Raum vor der Kamera positionierten oder bewegten

Infrarotquellen genutzt werden. LEE (2011) stellt auf seiner Internetseite verschiedene Anwendungen vor, die auf dieser Idee basieren, darunter beispielsweise auch ein Programm, mit dem eine kostengünstige Version eines Whiteboards hergestellt werden kann. Prinzipiell können damit mit Hilfe einer programmierten Gestenerkennung bestimmte Bewegungen identifiziert werden.

Mit Hilfe des Bluetooth Protokolls kann die Wiimote nicht nur mit der Wii Konsole kommunizieren, sondern lässt sich auch an jeden beliebigen Personal Computer als Human Interface Device, kurz HID, ähnlich wie andere PC-Peripheriegeräte anschließen. Alle Sensordaten werden mit bis zu 100 Hz übertragen und können damit auf dem Computer in Echtzeit verwendet werden (vgl. FRANZ UND SCHADER 2008). Welche dieser Daten und wie diese im Endeffekt verwendet werden, zeigt später Kapitel 7.3, das die Softwarearchitektur zeigt. Um über die verschiedenen Sensor- und Kommunikationstechniken einen Überblick zu bekommen, zeigt Abbildung 4.3 das Innenleben des Controllers, wobei auf der Top-Seite ebenfalls links neben dem A-Knopf der relativ kleine Beschleunuigungssensor zu sehen ist, auf der Bottom-Seite an der Spitze des Wiimote-Skeletts die Infrarotkamera zu erkennen sein sollte und auf derselben Seite weiter unten der Bluetooth Chip sitzt. Dieses komplette Technikpaket, das die Wii-Remote enthält, bietet sich wie in den folgenden Kapiteln zu sehen ist nicht nur für die Verwendung als Videospieleingabemodul an.

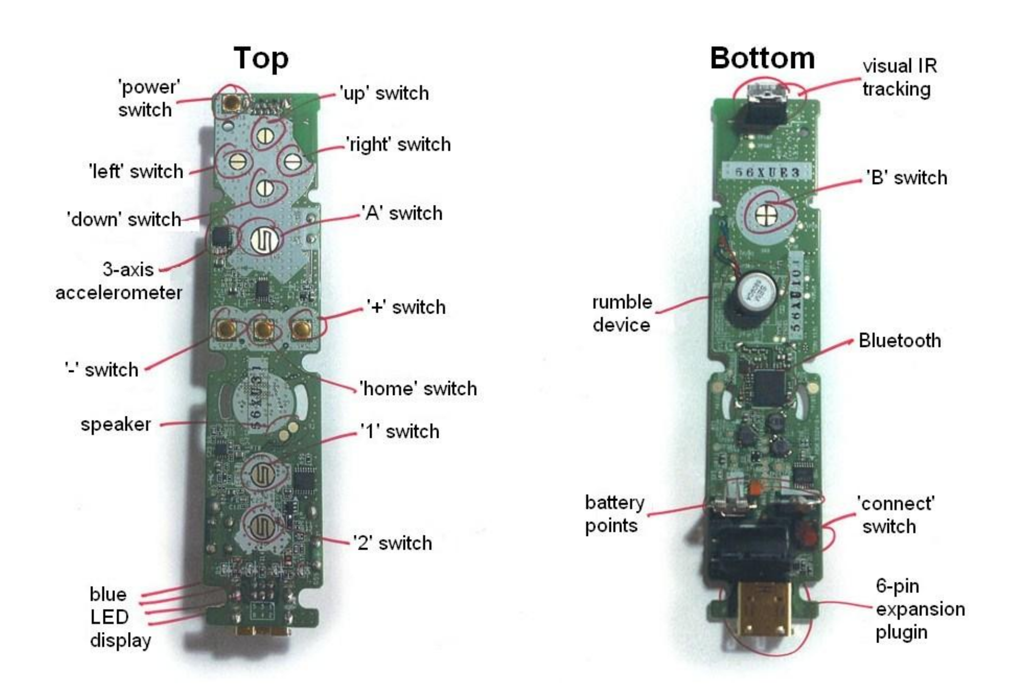


Abb. 4.3: Wiimote-Innenleben

4.2. Verwendete Datenübertragungsprotokolle

Wie bei vielen anderen Softwareanwendungen, wird beim Interfacedesign in dieser Arbeit auch auf bewährte Übertragungsprotokolle zurückgegriffen, die die Kommunikation zwischen den verschiedenen technischen Ausformungen des Systems ermöglichen soll. Dabei handelt es sich zum Einen um die drahtlose Funkverbindung mittels Bluetooth, die den Datenaustausch zwischen Wiimote und PC ermöglich und zum anderen um das schon öfters erwähnte MIDI-Datentransferprotokoll, mit dessen Hilfe die Interfacesoftware dem Klangerzeugungsprogramm die nötigen Befehle erteilt. Hierfür soll das folgende Kapitel einen Überblick geben.

4.2.1. MIDI-Schnittstelle

Das in ausformulierter Form korrekt bezeichnete Musical-Instrument-Digital-Interface wurde 1984 von führenden Syntesizer-Herstellern erfunden um einen Standard zum Datenaustausch zwischen elektronischen Instrumenten festzulegen und ist damit Teil des Digitalisierungs- und Virtualisierungsprozess der Musikproduktion (siehe Kapitel 3.1). Dieser Datenaustausch dient der Übermittlung von musikalischen Steuerinformationen wie Tonhöhe, Anschlagsstärke und Dauer des Tons. (vgl. MIDI.ORG 2011). Somit kann nach Abbildung 4.4 jedem MIDI-Befehl auch eine bestimmte Tonhöhe zugeordnet werden.

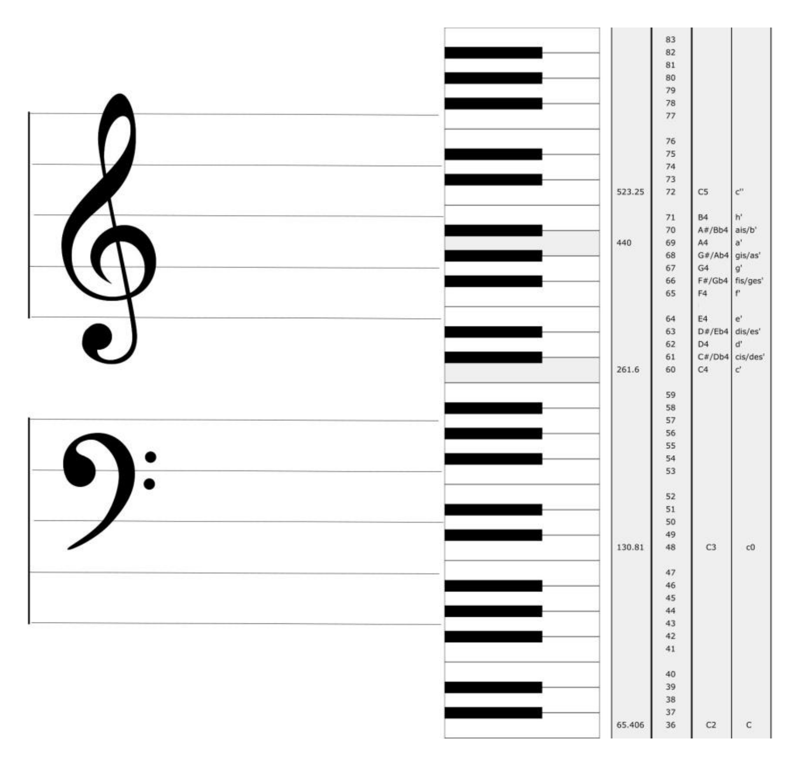


Abb. 4.4: MIDI-Notenwerte

Weiterhin gab es nach STANGE-ELBE durch die Einführung von MIDI „einen entscheidenden Schritt zur Teilung zwischen den die Musik konstituierenden elementaren Parametern (der Tonhöhe und Tondauer, der

Lautstärke und des Ausdrucks sowie des Tempos), also dem Spiel des Musikers und der Klangfarbe (den instrumentalen Eigenschaften)". Es ist nun beispielsweise mit einem MIDI-Keyboard möglich, alle Arten verschiedener Software Instrumente zu spielen. Eine Abfolge von verschiedenen MIDI-Noten, auch MIDI-Sequenz genannt, kann demnach in jedem Klangerzeuger interpretiert werden, wobei natürlich immer andere Töne produziert werden. Abbildung 4.6 zeigt in Logic Studio eine solche MIDI-Sequenz mit zehn Noten und daneben in Abb. 4.7 ihre Entsprechung in Notenschrift, die in den Digital Audio Workstations meist parallel dazu generiert wird.

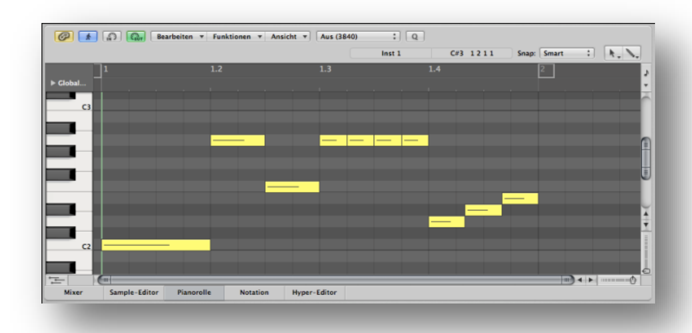


Abb. 4.6: MIDI-Sequenz in Logic

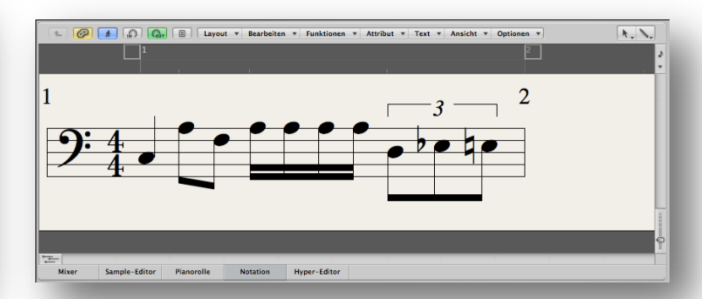


Abb. 4.5: Notenschrift Pendant

Im Gegensatz zu Audiodateien können die MIDI-Dateien aber nur in Verbindung mit einem virtuellen Instrument oder Synthesizer abgespielt werden.

Die Verbindung von Hardware erfolgt über eigene MIDI-Anschlüsse mittels der dazu passenden Verkabelung. Durch USB-Adapter können alle Geräte problemlos auch an den Computer angeschlossen werden (vgl. GÖRNE 2008).

Für die Interfaceentwicklung dieser Arbeit wird eine MIDI-Verbindung zwischen zwei Programmen gesucht. Zwischen dem MIDI-Sender, auch Output genannt, und dem –Empfänger (Input) muss immer eine vermittelnde Schnittstelle sitzen, die das Signal weiterleitet. Bei den Hardwaregeräten kann das etwa ein MIDI-Interface sein, das die MIDI-Hardwareschnittstellen besitzt und die Befehle über USB an den PC übergibt. Für eine computerinterne MIDI-Kommunikation eignet sich zum Beispiel das Freeware Programm LoopBe1, das kostenlos auf der Website NERDS.DE (2011) als Download zur Verfügung steht und auch in der dieser Arbeit beiliegenden CD enthalten ist. Dieser sogenannte MIDI-Treiber kann später MIDI-Befehle vom Interface empfangen, diese an den virtuellen Sampler weiterleiten und gestattet so die Softwarekommunikation. Prinzipiell können die MIDI-Daten aber natürlich auch an alle anderen Geräte weitergeleitet werden, die mit dem MIDI-Standard vertraut sind.

Daneben stellt sich nun die Frage nach der Konnektivität der Wiimote mit dem PC um dem Interface-Programm die notwendigen Daten überhaupt erst bereitzustellen.

4.2.2. Bluetooth-Verbindung

Naheliegend wäre es, da die Wii-Konsole im Normalfall den Bluetooth Standard verwendet, mit demselben Prinzip zu arbeiten. Hierfür ist in die Wiimote ein Bluetooth-Chip der Firma Broadcom eingebaut (siehe Abb. 4.3 aus Kapitel 4.1), mit dessen Hilfe eine Verbindung zu jedem Bluetooth-fähigen Computer hergestellt werden kann. Das Verbindungsprotokoll, das 1994 von der Firma Ericsson erfunden wurde, wird vor allem auf kurzen Strecken als Funktechnik eingesetzt und bietet mit einer Datenübertragungsrate von rund 2,1 Megabit pro Sekunde sowie Reichweiten von bis zu 100 Metern ein schnelles und verlässliches Kommunikationsprotokoll, auch für die Anwendung in dieser Arbeit (vgl. SAUTER 2008). Problematisch erscheint jedoch, dass die in vielen Personal Computern fest eingebauten Bluetooth-Empfänger nicht immer einwandfrei funktionieren. Darum kann zusätzlich noch ein USB-Bluetoothadapter verwendet werden, der dies garantiert. Im vorliegenden Fall wird zu diesem Zweck ein solcher Adapter der Firma Hama benutzt (siehe Abbildung 4.8). Ferner wird noch eine Software benötigt, die letzten Endes die Verbindung aufbaut. Eines dieser Programme, Blue Soleil, zeigt Abbildung 4.7. Das Programm zeigt mögliche Bluetooth-Kommunikationgeräte an und ermöglicht bei Bedarf den Datentransfer. An der Wiimote müssen für den Verbindungsaufbau die Knöpfe 1 und 2 betätigt werden, woraufhin die blauen LEDs der Fernbedienung anfangen zu blinken und ein Bluetooth-Signal ausgesendet wird, das von der Bluesoleil-Software erkannt wird und der Kontakt hergestellt wird.




Abb. 4.8: Hama Bluetooth-Adapter



Abb. 4.7: BlueSoleil-Software

4.3. Musiksoftware

Nachdem, durch Bluetooth und MIDI, die Kommunikationsmöglichkeiten soweit hergestellt sind, sodass Eingabebewegungen der IR-Quellen in die passenden MIDI Befehle übersetzt werden können, müssen nun folglich noch Programme zur Klangerzeugung ausgesucht werden. Wie bereits in der Konzeption vorgestellt, soll dafür ein Software-Sampler, der vor allem auf Drum- und Percussion-Sounds spezialisiert ist, eingesetzt werden. Grundvoraussetzung ist hierbei natürlich auch die Kompatibilität zum MIDI-Standard und daneben auch eine übersichtliche GUI, die das Auslösen der Samples und ihre Zuordnung zu den einzelnen MIDI-Befehlen anzeigt, damit die entwickelte Switch-Funktion sinnvoll genutzt werden kann. Eine weitere Bedingung ist, dass alle der verwendeten Musikprogramme kostenlos bezogen und verwendet werden können.

Darum wird die freie Demoversion des virtuellen Drumsamplers Battery 3 der Firma Native Instruments verwendet, der hier in Abbildung 4.9 in großem Format dargestellt wird. Hier liegen einzelnen Sounds auf den farbigen Pads der Software, die immer einer gewissen MIDI-Note zugewiesen sind. In dem Beispiel in Abb. 4.9 ist momentan das Sample „Cross Sticks C" in Zeile E und Spalte 8 ausgewählt, das der Note G5 entspricht. Der Notenwert kann zusätzlich durch Drücken der „Learn"-Taste auch einem anderen MIDI-Befehl zugeordnet werden. In den Standard-Einstellungen ist jeder Zeile eine Oktave und jedem der zwölf Pads in dieser Zeile wiederum einer der zwölf Halbtonschritte einer Oktave zugeordnet. Alle Samples können beliebig gewechselt werden, wobei nur unkomprimierte Audio-Formte wie .wav oder .aiff angenommen werden. Über das Einstellungsmenü lässt sich der LoopBe1 Treiber auswählen, der die Vermittlung der MIDI-Befehle von der Interfacesoftware aus bewerkstelligt (vgl. NATIVE-INSTRUMENTS.COM 2011). Zusätzlich kann



Abb. 4.9: Software-Sampler Native Instruments Battery 3

natürlich auch noch ein Sequenzing-Programm, wie z.B. eine Digital Audio Workstation verwendet werden, das Battery 3 dann als Plug-In über die VST-Schnittstelle einbindet (siehe Kapitel 3.1.2) und die in der Lage ist, die MIDI-Daten aufzuzeichnen.

Im Internet finden sich dazu zahlreiche Angebote, die entweder Demo-Versionen von professionellen Audioversionen oder funktionstechnisch sehr eingeschränkte DAWs darstellen. Zu einer der Letztgenannten zählt Mulab 3, das auf MUTOOLS.COM (2011) als kostenfreier Download verfügbar ist. Abbildung 4.10 zeigt das Programm, das die geforderten Bedingungen erfüllt, indem es fähig ist virtuelle Instrumente als Plug-Ins einzubinden und darüber eingespielte MIDI-Sequenzen aufzuzeichnen. Wie die Benutzung der Musiksoftware im Einzelnen vor sich geht wird durch Kapitel 7.3.3.2 im Kontext der Anleitung des Gesamtprogramms erläutert.



Abb. 4.10: Mutools Mulab 3

Auf der CD, die der Bachelor-Arbeit beiliegt, befinden sich zur Installation auch die Setup-Dateien beider Programme.

5. Konzeption

In diesem Abschnitt werden die Ergebnisse und Zusammenhänge des Prozesses der Ideenfindung dargestellt und in verschiedenen Konzepten präsentiert. Die genauen technischen Details und Ausführungen werden erst in Kapitel 7 zur Softwareprogrammierung dargelegt. Es geht vielmehr darum, einen Überblick über das entwickelte System zu gewinnen. Daher sind an vielen Stellen auch Verweise zu anschließenden Kapiteln eingefügt, um den Charakter einer Übersicht zu wahren. Allerdings muss an dieser Stelle auch erwähnt werden, dass sich viele der nun folgenden Erkenntnisse und Überlegungen erst während des Programmier- und Entwicklungsvorgang ergeben haben.

Die wichtigste Eigenschaft, die bereits definiert wurde und das System charakterisieren soll, ist die intuitive Befehlseingabe über die Wiimote. Daher ist es eine wichtiges Anliegen dieses auch zu garantieren. Im Sinne der Definition von „intuitiv" in Kapitel 2.1 in Bezug auf ein Interfacesystem kommt es demnach darauf an, dass unbewusste psychische Prozesse des Handelns genutzt werden können, um sie in den Vorgang der Erzeugung von Musik einzubinden. Das kann beispielsweise die direkte Reaktion auf ein auditives Signal sein, auf das der Nutzer antwortet oder Bewegungen im Rhythmus, die als Eingabe verwendet werden. Die Bedienung soll wenn möglich auch eine natürliche Geste sein, die um eine schnelle Eingewöhnung in das System für die verschiedensten Gruppen von Nutzern sicherzustellen und eben den Bezug zur klassischen Form des Musizierens und der Körperlichkeit, die in Kapitel 3.2 eingehend beschrieben wurde, herzustellen. Zusätzlich müssen diese Eingaben vom System und damit für die Software immer sicher und am besten auch schnell zu erkennen und in das musikalische Ergebnis umzusetzen sein. Hierfür ist das zu schreibende Programm in Visual Studio C# zuständig, das den Software-Teil des Projekts darstellt (siehe Kapitel 7.3).

Zusätzlich soll auch das universelle MIDI-Format verwendet werden, mit dessen Hilfe Interfacesoftware und virtueller Klangerzeuger kommunizieren. Ziel des Einsatzes dieser Schnittstelle ist eine möglichst vielfältige Verwendung des Interfaces, mit der alle Arten virtueller Musiktechnik angesteuert werden können wie in Kapitel 4.2.1 erklärt wurde.

Für die Aufnahme der Eingabe kommen dabei die zwei in Kapitel 4.1 beschriebenen Arten der Sensorik der Wii-Remote in Frage. Die Bedienung über Tasten und Sticks wird hierbei außer Acht gelassen, da diese definitiv hinsichtlich der bereits oben erwähnten intuitiven Bedienung weitaus weniger interessant sind als die Verwendung der Beschleunigungssensoren oder der Infrarotkamera der Wiimote.

Hinsichtlich der Entscheidung des verwendeten Klangerzeugers werden ebenfalls nur zwei Möglichkeiten in Betracht gezogen, die mit den Sensortechniken der Wiimote angebracht gesteuert werden können.

Bei der Nutzung eines Synthesizers als Musikproduktionsprogramm könnten verschiedene Parameter, die den Klang modulieren, entweder durch die Lage im Raum mittels der Beschleunigungssensoren oder die Position der Infrarotquellen durch die Wiimote- Kamera gesteuert werden. MIKE.VERDONE.CA und MOORE zeigen allerdings bereits, die Umsetzung einer solchen Idee aussehen.

Falls der Samplers als Form des virtuellen Instruments eingesetzt wird, gleicht die Eingabe eher einer Art imaginären Schlagzeug. Durch abrupte schlagähnliche Bewegungen, die entweder mit der Wiimote selbst

ausgeführt werden könnten, um durch den Beschleunigungssensor detektiert zu werden, oder mit Infrarotsignalen, deren Bewegung wiederum mit Hilfe der Wiimote-Kamera aufgenommen und analysiert werden. WIIDRUMSYNTH.CODEPLEX.COM zeigt für die Verwendung der Beschleunigungssensoren in diesem Fall bereits ein Beispiel.

Eine Kombination beider Sensortechniken kommt aus trivialem Grund nicht in Frage. Grundsätzlich ist das System auf nur einen Nutzer ausgelegt. Daraus folgt dass durch diesen Anwender entweder das Infrarot-Signal bewegt wird oder die Wiimote, aber nicht beides gleichzeitig geschehen kann. Für die Verwendung der Beschleunigungssensoren muss die Fernbedienung allerdings dynamisch beansprucht werden und wird somit entweder relativ schnell bewegt oder um die verschiedenen eigenen Achsen gedreht. Bei beiden dieser Anwendungsarten kann ein ruhendes Infrarotsignal durch die Bewegung der Wiimote wenn überhaupt nur sehr umständlich anvisiert und genutzt werden. Andererseits sollte bei einer Bewegung der Infrarotquelle die Wii-Remote in Ruhe bleiben um das Signal angebracht verarbeiten zu können wobei wiederum die Beschleunigungssensoren nicht verwendet werden können. Demzufolge scheint eine Beschränkung auf eine der beiden Sensortechniken sinnvoll.

Da dieses Bachelor-Projekt den Anspruch besitzt eine vollkommen neuartige Idee umzusetzen, die in dieser Form, zumindest hinsichtlich den Recherchen, die während der Arbeit gemacht wurden, noch nicht umgesetzt worden ist, fällt die Entscheidung letztlich zugunsten eines Samplers der über die Infrarottechnik der Wiimote bedient wird und dessen Konzeption im Folgenden beschrieben werden soll.

5.1. Funktion

An erster Stelle steht hierbei natürlich der Gedankengang welche Funktionen, die auch hinsichtlich der programmiertechnischen Realisation zu bewerkstelligen sind, das System erfüllen soll. Dafür scheint ein Blick auf den Aufbau und die Eigenschaften des zu spielenden Software-Samplers sowie auch seinen Hardware-Vorgängern sehr empfehlenswert. Dabei handelt es sich um eine spezielle Art von Sampler, die im Wesentlichen zwar auch dazu fähig sind instrumentale Samples abzuspielen, aber im Allgemeinen eher für perkussive und Drum-Sounds in der Musikproduktion zuständig ist.



Abb. 5.1: Battery 3



Abb. 5.3: AKAI MPC 1000



Abb. 5.2: NI Maschine

In den drei Abbildungen 5.1 bis 5.3 sind verschiedene Arten von Samplern zu erkennen. Zum einen zeigt 5.2 mit dem MPC 1000 der Firma Akai einen kompletten Hardware Sampler und Nachfahren des in Kapitel 3.1.1 vorgestellten MPC 60. Desweiteren sind in 5.1 und 5.3 die reine Sampler-Software Battery 3 und ein Controller für derartige Programme, die sogenannte Maschine, beide von Native Instruments, zu sehen. Die Gemeinsamkeit dieser Produkte sind die quadratischen Pads auf ihrer Oberfläche, die mit verschiedenen Samples belegt sind, welche sich durch eine Druckberührung der Pads abspielen lassen. Vor allem für die deswegen meist verwendeten Drum- und Percussion-Sounds eignet sich diese Technik hervorragend um im Rhythmus des Musikstücks verschiedene Samples einzuspielen. Aus diesem Grund wird auch dem hier vorgestellten Interface genau dieses Funktionsprinzip in ähnlicher Weise zugrunde liegen.

Hierzu soll das Fenster in dem die Wiimote Infrarotsignale wahrnehmen kann, in rechteckige annähernd gleichgroße Flächen unterteilt werden, dem dann jeweils andere Samples, wie bei den gezeigten Beispielen, zugeordnet werden. Die Betätigung der Sounds soll durch ruckartige schnelle vertikale Bewegung in den beschriebenen Feldern erfolgen. Nach eigenen Messungen kann die Infrarotkamera Signale in einem Winkel von ungefähr 40 Grad in horizontaler Richtung und 30 Grad in vertikaler Richtung wahrnehmen (siehe Abb. 5.4). Demnach stellt bei einem Abstand der Infrarotquelle zu Kamera von einem Meter, die Arbeitsfläche ein Rechteck mit den Seitenlängen von 0,72 und 0,54 Metern dar. Dies entspricht mit einem Seitenverhältnis von 4:3 auch logischerweise der Auflösung der Kamera von 1024x768 Pixel.

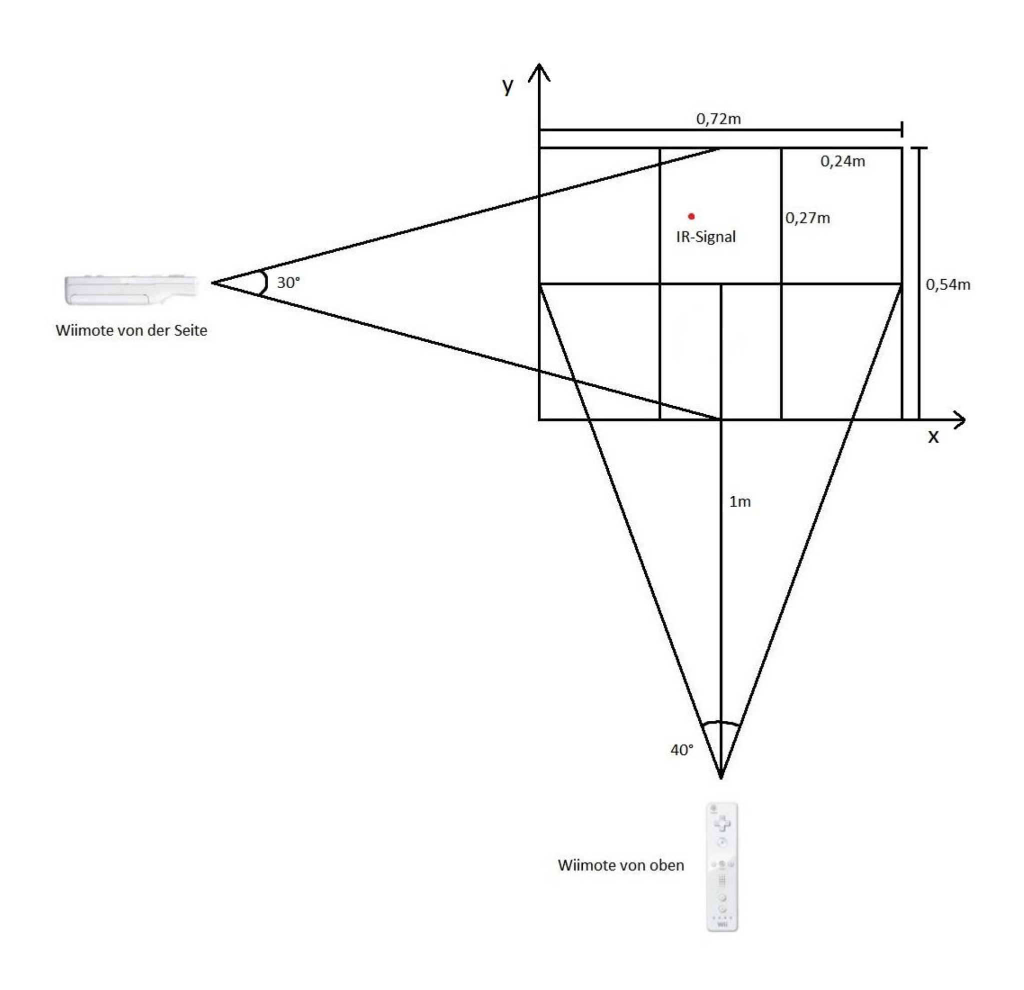


Abb. 5.4: Wahrnehmnung der Infrarotkamera und Einteilung der Arbeitsfläche

Um ausreichend Spielraum für die vertikale Bewegung zum Auslösen der Samples zu gewährleisten, müssen nun die einheitlichen Teilflächen dimensioniert werden. Da sich die Distanz von ungefähr einem Meter zwischen Wiimote und IR-Quelle im Laufe der weiteren Entwicklung als sinnvoll herausgestellt hat, wird die imaginäre Arbeitsfläche des Interfaces nun in insgesamt sechs gleichförmige Bereiche unterteilt, die, wie Abb. 5.4 ebenfalls darstellt, mit den Ausmaßen von 0,27m mal 0,24m annähernd quadratische Formen annehmen und sich in zwei Felder Richtung der y-Achse und drei Felder bezüglich der x-Achse gliedern. Mit der soeben erfolgten Aufteilung, steht genügend Platz zur Verfügung um die Betätigungsbewegung einwandfrei auszuführen.

Zunächst sei hier erwähnt, dass, um die exakte Kontrolle der IR-Signale zu garantieren, Handschuhe mit darauf befestigten Infrarot-Leuchtdioden verwendet werden, die in Kapitel 6 vorgestellt werden. Von den insgesamt drei LEDs sitzen zwei auf beiden Zeigefingern der Handschuhe und der dritte auf dem rechten Mittelfinger. Ziel dabei ist es mit den Zeigefinger-Dioden die Betätigungsbewegung auszuführen, ähnlich den Bewegungen eines Schlagzeuger, nur ohne den Stockeinsatz. Die dritte LED wird zum Ausführen des sogenannten Switch-Befehls benötigt, der weiter unten beschrieben wird.

Hiernach stellt sich noch die Frage wie diese Geste zum Auslösen der Samples erkannt werden soll. Eine wichtige Überlegung ist hierbei, durch ein- bzw. zweifache Differentiation der Positionsdaten der IR-Signale zusätzlich noch auf deren Geschwindigkeits- und Beschleunigungswerte zugreifen zu können. Im weiteren Verlauf der Arbeit zeigt sich, dass sich für diese Art Gestenerkennung die Verwendung des Beschleunigungswerts in negative y-Achse bezüglich der Abbildung 5.4 anbietet. Das bedeutet konkret, dass falls dieser Beschleunigungswert einen gewissen Schwellenwert überschreitet, ein MIDI-Befehl zum Betätigen des Feldes und Abspielen des Sounds gegeben wird. Darüber hinaus soll ein visuelles Feedback eingebaut werden, das dem Nutzer signalisiert ob und in welchem Feld der Ton letztlich ausgelöst worden ist.

Weiterhin soll das System ebenfalls die Funktion des Switch-Befehls unterstützen, sodass mit einer anderen Eingabebewegung die auf den Pads liegenden Samples wechseln. Dafür soll der MIDI-Parameter der Tonhöhe, der dem Feld zugewiesen ist, geändert werden, da dieser in vielen Software-Samplern auch die Ansteuerung der einzelnen Samples beschreibt. Um auszuschließen, dass mit dieser Geste die Betätigungsfunktion ausgelöst wird, ist der Switch-Befehl etwas komplexer und benötigt alle drei LEDs. Mit der linken Hand, an der nur eine Infrarotquelle (IR 1) sitzt, wird durch die sogenannte Auswahlbewegung auf das Feld gezeigt, in dem das Sample gewechselt werden soll. Die anderen zwei IR Signale (IR 1 und IR 2), die an der rechten Hand befestigt sind, müssen gleichzeitig und untereinander, also mit ähnlicher x-Koordinate, eine schnelle horizontale Bewegung, die Auslösebewegung, ausführen, die aber in keinem Bestimmten der Felder, sonder nur im Aufnahmebereich der Kamera stattfinden muss. Geht diese Bewegung in negative Richtung der x-Achse, so wird der MIDI-Befehl der Tonhöhe einem Halbton entsprechend nach unten korrigiert und die gesamte Anweisung als Switch-Back-Befehl bezeichnet, bei der Bewegung parallel zur positiven x-Achsenrichtung erfolgt die Korrektur dementsprechend nach oben und wird mit dem Namen Switch-Forward-Befehl versehen. Abbildung 5.5 zeigt eine Art den Switch-Back-Befehl im linken, oberen Bereich auszuführen.

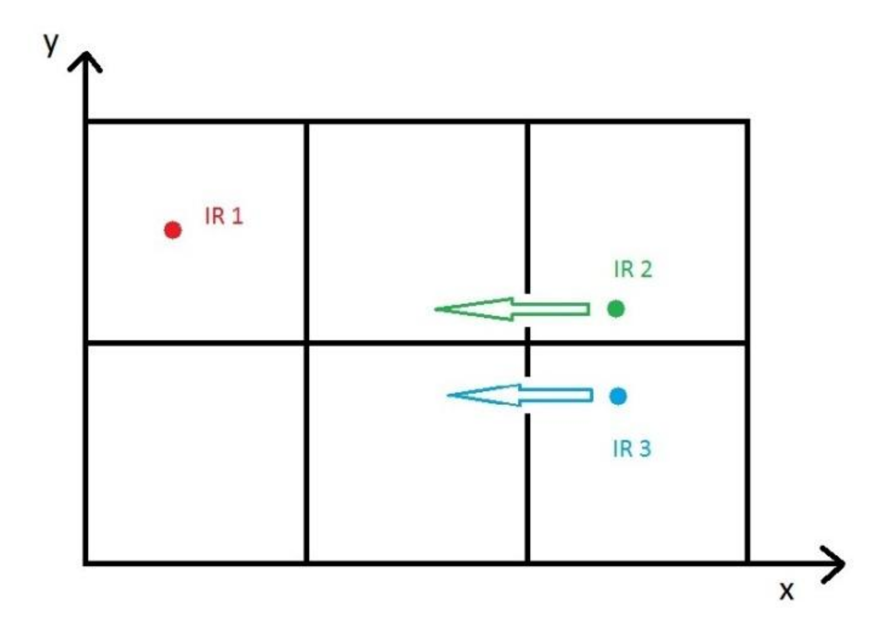


Abb. 5.5: Switch-Back-Befehl

Mehr als diese beiden Funktionen, die jedoch die wichtigsten Grundprinzipien eines Hardware-Samplers oder eines Softwaresampler-Controllers darstellen, soll das zu entwickelnde Interface zunächst nicht erfüllen. Dafür verfügt es aber, indem es nur über die Bewegung der Finger (mittels LED-Handschuhe) bedient werden kann, über eine sehr intuitive Art der Eingabeform, die letztlich zur Erzeugung eines Tons führt. Des Weiteren kann, wie bereits geschildert, jede MIDI-kompatible Software als Klangerzeuger genutzt werden. Hinsichtlich der Anwendung mit den zu steuernden Programmen ist das Interface also flexibel und universell einsetzbar. Ein weiterer Vorteil besteht in der Individualisierbarkeit der Kombination der einzelnen auf den Feldern liegenden Samples. Der Anwender kann so lange mit Hilfe des Switch-Befehls die einzelnen Sounds wechseln, bis ihm eine passende Zusammenstellung vorliegt, mit der er anschließend arbeitet. Dieses Prinzip von Individualisierbarkeit entspricht daher in gewissem Maße auch den Richtlinien von nutzergerechter Gestaltung bezüglich Usability (Kapitel 2.2).

5.2. Software-Bedienoberfläche

Nachdem die grundlegenden Funktionen des Systems definiert und erklärt wurden, gilt es nun die Anforderungen an die grafische Benutzeroberfläche, kurz GUI (Grafical User Interface), zu stellen. Obwohl auch hier ein kleiner Vorgriff auf das die Programmierung betreffende Kapitel geschieht, sind viele Ideen zur GUI schon bereits während der Konzeption entwickelt worden und dienen zum Verständnis des Gesamtsystems. Zusätzlich wird darauf hingewiesen, dass auch einige der auf der grafischen Benutzeroberfläche vorhandenen Objekte Funktionen des Interfaces sind, sie aber durch ihre Zugehörigkeit zur GUI erst in diesem Kapitel thematisiert werden. Die detaillierte Betrachtung der Softwareoberfläche, die

mit Abbildung 6.5 auf Seite 48 auch das endgültige Aussehen des GUI zeigt, wird erst in Kapitel 7.3.1 vorgenommen. Hier werden zunächst nur die Ansprüche an die Systemoberfläche beschrieben.

Elementar beim Entwurf der Benutzeroberfläche ist die Frage, welche Information über den Status und die Aktivität des Systems relevant sind und damit angezeigt werden müssen um entweder die Interaktion mit der Schnittstelle überhaupt erst zu ermöglichen oder dem Nutzer bei der Anwendung behilflich zu sein. Dabei werden hier aber nur die wichtigsten Grundfunktionen der GUI abgedeckt. Insgesamt ist das Softwaredesign natürlich nicht mit professionellen Markenprodukten vergleichbar. Zu diesen soeben erwähnten fundamentalen Funktionen, die zur Ausgabe von Informationen gehören, ist die Positionsanzeige der Infrarotsignale wahrscheinlich die wichtigste. Um sich im Raum vor der Infrarotkamera zurechtzufinden und um zu wissen in welchem der Sample-Feldern sich die IR-Dioden gerade befinden, ist die Rückgabe dieser Positionsangaben unabdinglich. Wichtig ist dabei auch die Unterscheidung der für das System relevanten maximal drei verschiedenen IR-Quellen. Hierbei empfiehlt es sich die Signale auf der GUI mit unterschiedlichen Farben zu kennzeichnen, die jedoch klar erkennbar sein müssen. Abbildung 5.5 stellt bereits einen ersten Entwurf der Rückgabefläche der IR-Positionsdaten dar.

Daneben verschafft die simple Information, ob eine Wiimote (und mit ihr die benötigte IR-Kamera) überhaupt angeschlossen ist, Klarheit ob das System funktionsfähig ist. Dies wird später kombiniert mit der Anzeige der restlichen Batterieleistungsfähigkeit, die zudem auf eine verbleibende Restverwendungszeit der Wiimote und somit des Gesamtsystems schließen lässt.

Weiterhin soll eine Regelungsmöglichkeit in die Softwareoberfläche eingebaut, mit dessen Hilfe die Sensitivität hinsichtlich der Aufnahme der IR-Bewegungen eingestellt werden kann. Hierdurch soll eine Anpassung der Systemkonstanten erfolgen, die die Funktion des Systems auch dann gewährleistet wenn der Anwender das Interface mit einem größeren Abstand als einen Meter zur Wiimote-Kamera nutzen will.

Für die Ende des letzten Kapitels beschriebene Möglichkeit des Kombinierens der gewünschten Samples, ist es zudem von Vorteil wenn durch ein grafisches Element verdeutlich wird, welche Sounds momentan auf den verschiedenen Feldern liegen oder zumindest welche MIDI-Befehle, denen in der Sampler-Software Sounds zugeordnet werden können, durch das Auslösen der Pads gesendet werden.

Um über die MIDI-Schnittstelle aber überhaupt kommunizieren zu können, muss als letzter Punkt der Anforderungen an die Softwareoberfläche noch ein Einstellungsmenü in die Benutzeroberfläche eingebaut werden, worüber sich anpassen lässt, welcher MIDI-Treiber, der letztlich zwischen Interface und Musiksoftware vermittelt (siehe Kapitel 4.2 und 4.3), verwendet wird.

Insgesamt scheint der Anspruch an die GUI im Vergleich zu manch anderem Programm relativ klein, es sollte mit der Umsetzung dieser Bedingungen jedoch möglich sein, problemlos an der Interaktion mit dem System teilzunehmen.

6. LED-Handschuhe

Bevor die Beschreibung des Erstellungsprozesses der Softwrae, soll nun noch kurz auf die in eigener Arbeit gefertigten LED-Handschuhen eingegangen werden. In Abbildung 6.1 und 6.2 sind aus Gründen der Veranschaulichung zunächst die Handschuhe von der Vorder- und der Draufsicht zu sehen.



Abb. 6.1: LED-Handschuhe (Draufsicht)



Abb. 6.2: LED-Handschuhe (Vorderansicht)

Die insgesamt drei Dioden sitzen, wie bereits genannt, an der rechten Hand an der Spitze von Zeige- und Mittelfinger und am der linken Hand nur am Zeigefinger. Dabei handelt es sich um T-1 Infrarot-LEDs der Firma Kingbright mit einem Durchmesser von 3,2mm, die eine Spitzenwellenlänge von 940 Nanometer emittieren und zwischen 1,2 und 1,6 Volt arbeiten. Für weitere technischen Details sei auf das Datenblatt in Anhang A verwiesen. Die notwendige Spannung stellt jeweils eine AAA Micro-Batterie mit einer Spannung von 1,6 Volt zur Verfügung, die in passenden Halterungen, angeklebt an die Handschuhe, eingelegt werden. Da aber insgesamt durch Widerstände in Batteriehalterung und Kabel nur ein kleiner Teil der Gesamtspannung abfällt, steht den Dioden auch an der rechten Hand, an der zwei davon parallel geschaltet sind, eine für ihren Arbeitsbereich optimale Voltzahl zur Verfügung. Verbunden sind alle Komponenten durch in eigener Arbeit angelötete Verbindungskabel. Zur Befestigung der LEDs ist Heißkleber verwendet worden. Insbesondere in Abbildung 6.2 sind die leuchtenden Dioden von vorne zu erkennen, da handelsübliche Digitalkameras auch Infrarotstrahlung bis zu einem gewissen Grad abbilden. Zunächst wurde zusätzlich noch ein Schalter in den Stromkreis mit eingebaut werden, von dem aber eine massiver Wackelkontakt ausgelöst wurde, sodass er wieder entfernt werden musste. Die LEDs werden nun letztlich dadurch an- und ausgeschaltet, indem die Batterien eingelegt bzw. entfernt werden.

Mit diesem Kapitel wird der Teil von Konzeption und Hardware beschlossen. Es folgt die Ausarbeitung, die den Prozess der Softwareerstellung und ihrer Erklärung beleuchtet.

.

7. Softwareprogrammierung

Nach dem Abschluss der Konzeption besteht die Hauptaufgabe zur Entwicklung des Projekts in der Erstellung der Interface-Software. Hierzu sind in den vorigen Kapiteln bereits die Anforderung an Funktion und Oberflächen des Programms definiert worden und müssen nun umgesetzt werden. Wie in Kapitel 5 bereits erwähnt, wird für diesen informatischen Teil der Arbeit die Programmierumgebung Visual Studio Express 2010 von Microsoft mit der Sprache C# verwendet. Darum soll in den folgenden Abschnitten zur Softwareprogrammierung zunächst eine Begründung für die Auswahl von Visual Studio und C# gegeben werden bevor anschließend die Verwendung der Managed Libraries erklärt wird, mit denen auf die Parameter der Wiimote zugegriffen werden kann, um daraufhin das eigentlich geschriebene Programm, dem der Name „Wii Infrared MIDI" gegeben wurde, in aller Ausführlichkeit zu beschreiben.

7.1. Visual Studio Express

Die integrierte Entwicklungsumgebung Microsoft Visual Studio 2010 unterstützt diverse sogenannter Hochsprachen zum Erstellen eines Programmcodes. Neben dem verwendeten C# sind noch Visual Basic, C++, C, C++/CLI und F# zu nennen, von denen alle auf dem .NET Framework basieren, einer von Microsoft entwickelten Softwareplattform.

Dieses besteht aus einer Laufzeitumgebung für die Ausführung der erstellten Programme, diversen Programmierschnittstellen und zusätzlich noch eine Sammlung von selbstständigen Programmkomponenten, sozusagen einer Bibliothek für standardmäßig und oft verwendete Befehle, die in der Softwareerstellung eingesetzt werden. Darauf ist auch der erste zu nennende Vorteil der Plattformunabhängigkeit zurückzuführen. Prinzipiell wird bei der Programmierung mit .NET der Code, nach seiner Ausarbeitung in der gewählten Sprache, in einen sogenannten Zwischencode namens Common Interface Language (CIL) übersetzt.

Dieser ist unabhängig vom Prozessor, kann vom ihm aber in seiner Form auch nicht ausgeführt werden. Dazu wiederum ist ein Compiler von Nöten der die Universalsprache CIL für den Prozessor übersetzt. In der .NET Softwareplattform handelt es sich dabei um die Laufzeitumgebung Common Language Runtime (CLR). Die gesamte Systematik wird in Abbildung 7.1 dargestellt. Die Flexibilität bezüglich der Plattform kommt nun daher, dass die CIL mit dem passenden Compiler auch für andere Zielsysteme wie Mac OS X, Linux und weitere übersetzt werden kann (vgl. KÜHNEL 2010).

Zusätzlich zu diesem Vorteil ist zu erwähnen, dass Visual Studio Express Produkte kostenfrei auf der zugehörigen Website zu beziehen sind und auch ohne Einschränkungen auch für kommerzielle Projekte verwendet werden können. Darüber hinaus, und dabei handelt es sich um einen der Gründe warum die Wiimote überhaupt als Eingabemedium ausgesucht wurde, sind für C# bereits sogenannte Managed Libraries (siehe nächstes Kapitel) vorhanden, mit dessen Einbeziehung es überhaupt erst möglich wird direkt auf die benötigten Parameter der Wiimote zuzugreifen. Im weiteren Verlauf dieses Kapitels soll nun noch in sehr kompakter Form auf den prinzipiellen Aufbau der Codestruktur eingegangen werden um damit die

Vorstellung des eigentlichen Programmcode in Kapitel 7.3.2.4 möglicherweise besser nachvollziehen zu können.

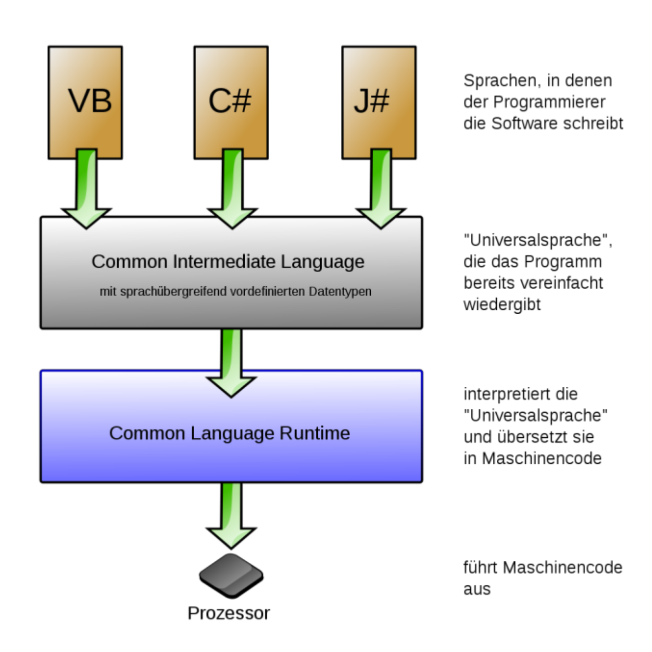


Abb. 7.1: Basisprinzip von CIL und CLR

7.1.1. Programmiersprache C#

Der C#-Code zählt zu den objektorientierten Programmiersprachen. Ein Objekt ist eine Sammlung zueinander in Beziehung stehender Informationen und Funktionen. Das kann etwas sein, das über ein entsprechendes Äquivalent in der tatsächlichen Welt verfügt, das virtuelle Bedeutung hat wie ein Fenster oder andere grafische Elemente auf der Benutzeroberfläche oder es kann einfach ein abstraktes Element innerhalb eines Programms darstellen. Es bezeichnet dabei immer ein Exemplar eines bestimmten Datentyps, der in diesem Zusammenhang auch als Instanz einer Klasse bezeichnet wird. Jedes Objekt kann durch verschiedene Attribute beschrieben werden und verschiedene Zustände annehmen und diese auch auslösen. Eine Klasse dagegen, die in einer objektorientierten Sprache meist den Mittelpunkt der Anwendung darstellt, beschreibt umgangssprachlich formuliert ein Modell von Objekten und ist diesen übergeordnet. Sie besitzt Variablen, die bestimmte Eigenschaften beschreiben, Methoden, die Tätigkeiten darstellen und Ereignisse, die die Folge von Zuständen sind bzw. diese auslösen. Innerhalb der Methoden erfolgen dann meist die sogenannten Operationen wie beispielsweise die bedingte *if-else*-Anweisung oder etwa Kontrollstrukturen wie die *for-* und *while*-Schleife. Die Klasse wiederum ist immer einem bestimmten Namensraum, meist Namespace genannt, untergeordnet. Dieser stellt sozusagen die Namen aller Objekte, mit dem diese aufgerufen und identifiziert werden können, in einer Art Baumstruktur zur Verfügung. Daneben definiert er auch die den Objekten zugeordneten Eigenschaften und Methoden. Der komplette Namespace kann auch durch Verweise auf „Assemblys“ erweitert werden, welche typischerweise in.dll- oder .exe-Dateiform bereitgestellt werden.

Die Managed Libraries wie die Wiimote Library bezeichnen ebenfalls „Assemblys“, auf die in Visual Basic dann referenziert werden kann. (vgl. GUNNERSON 2000)

7.1.2. Windows Forms Application

Bei der Arbeit mit Visual Studio gilt es zunächst auszuwählen, welche Art einer Anwendung programmiert werden soll. Neben mehreren anderen Möglichkeiten wird auch die Programmierschnittstelle „Windows Forms“ angeboten, die gerade zur Erstellung windows-ähnlicher, grafischer Benutzeroberflächen geeignet ist. Hier wird dem Programmierer neben dem normalen Fenster für die Code-Eingabe noch ein sogenannter Designer zur Verfügung gestellt, dessen Ansicht Abb. 7.2 zeigt und mit dem die standardisierten Schaltflächen und Elemente, die aus Windows Anwendungen bekannt sind, durch einfaches Drag & Drop aus der Toolbox auf der linken Seite der Ansicht in die Software-Oberfläche integriert werden können. Desweiteren wird rechts noch ein Menu zur Bearbeitung der Eigenschaften aller Bestandteile angeboten. Alle hier eingebauten Schaltflächen mit deren Eigenschaften erscheinen automatisch im Programmcode, in dem diese wahlweise auch editiert werden können. (vgl. KÜHNEL 2010)

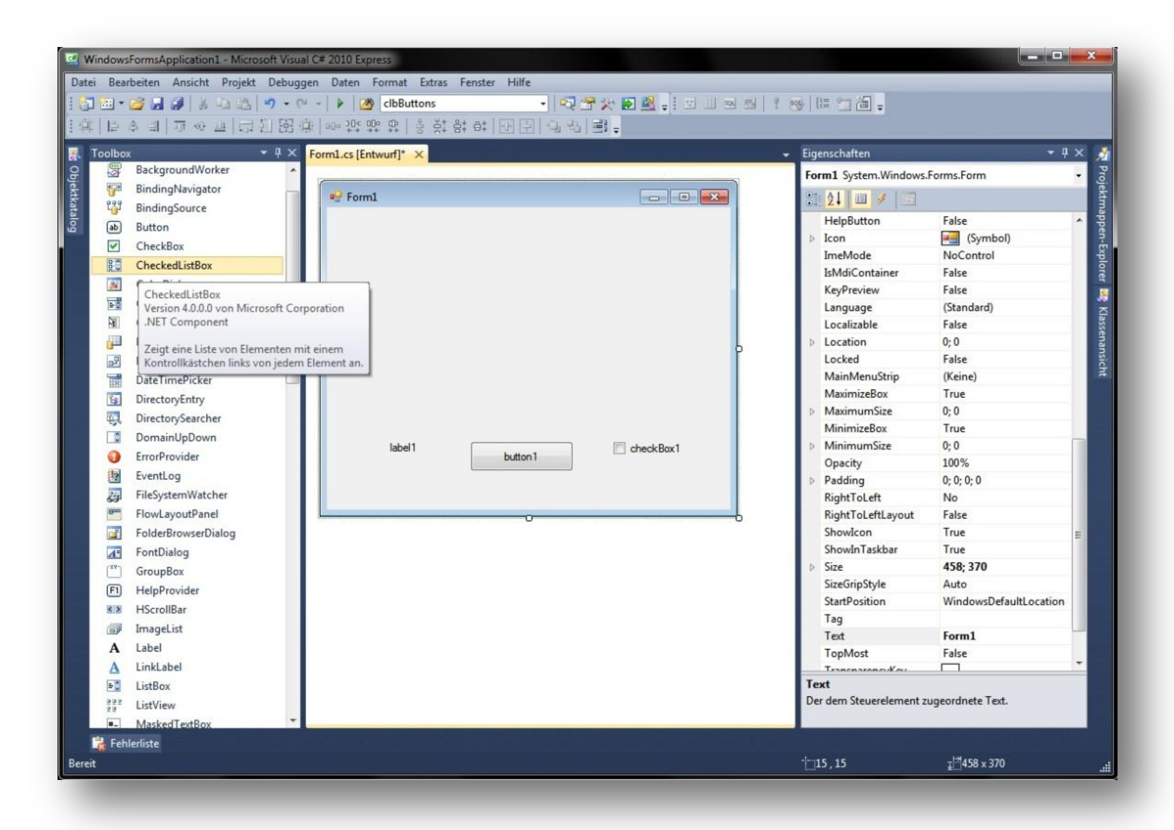


Abb. 7.2: Visual Studio 2010 in der Designer Ansicht

Zusammengefasst vereint Windows Forms also die Eigenschaften einer für Einsteiger in die Welt des Softwareengineerings durchaus strukturierten und visuell anschaulichen Oberfläche, der daraus resultierenden relativ leichten Handhabung uns ist zusätzlich noch auf die explizite Auslegung von GUI-Erstellung spezialisiert. Daher wird dieser Programmierschnittstelle auch für die Interfacesoftware Wii Infrared MIDI verwendet.

Diese kurze Erklärung erscheint womöglich sehr abstrakt und undurchsichtig, wird aber hoffentlich klarer, wenn die Beschreibung des Programmcodes von Wii Infrared MIDI erfolgt. Eine ausführlichere Darstellung und Erklärung der allgemeinen Verwendung der Codestruktur für C# würde aber definitiv den Rahmen dieser Arbeit sprengen. Dazu wird auf GUNNERSON (2000) verwiesen.

7.2. Managed Libraries

Die enorme Relevanz dieser Libraries ist im vorigen Kapitel bereits angedeutet worden. Das Wort „Managed" beschreibt im Endeffekt nur das Basieren der Verzeichnisse auf dem .NET Framework, sodass Visual Studio diese verarbeiten kann. Neben der bis jetzt schon des Öfteren benannten WiimoteLib von PEEK (2009), die im Rahmen des CodePlex Open Source Porject auf CODEPLEX.COM (2011) heruntergeladen werden kann, wird zusätzlich noch das C# MIDI Toolkit von SANFORD (2007) verwendet, das ebenfalls kostenlos bezogen werden kann, wodurch es in Visual Studio möglich wird alle nötigen MIDI-Befehle zu programmieren. In der WiimoteLib wird dem Konzept entsprechend vor allem auf die Daten der Infrarotkamera zugegriffen, speziell auf die Positionen und Eigenschaften der einzelnen IR-Signale, während im C# MIDI Toolkit die Variablen des MIDI-Notenwerts und der –Velocity benötigt werden sowie die simple Anweisung ob ein MIDI-Befehl ausgesendet wird. Alle diese Vorgänge werden im Kapitel zu eigentlichen Softwareimplementierung ausführlicher dokumentiert. Es sei noch ein Hinweis auf die Vielzahl an eindrucksvollen Anwendungen gegeben, die gerade auf der WiimoteLib basieren und auf WIIMOTE BASED APPLICATIONS (2011) angeschaut werden können.

7.3. Wii Infrared MIDI

Der folgende Teil kann als der Hauptteil dieser Bachelor-Arbeit bezeichnet werden, denn im Folgenden wird das Interface Programm detailliert beschrieben. Dazu wird zunächst auf die Software-Bedienoberfläche eingegangen, die nach Möglichkeit den in Kapitel Konzeption verfassten Anforderungen entsprechen sollte. Desweiteren wird im Abschnitt zur Implementierung in aller Einzelheit der Aufbau von Wii Infrared MIDI mit den einzelnen Methoden bezogen auf deren Funktionen beleuchtet. Auch hier sollen dementsprechend die formulierten Ansprüche umgesetzt werden. Abschließend soll für den Anwender noch eine Installations- und Gebrauchsanweisung gegeben werden, die jedem Nutzer den Einsatz von Wii Infrared MIDI anschaulich näherbringen sollte. Alle programmtechnischen Begriffe sind in der dem Quelltext entsprechenden Schrift im Text eingebaut und dazu zusätzlich zur Erkennung kursiv geschrieben.

7.3.1. Graphical User Interface

Die Softwareoberfläche von Wii Infrared MIDI ist, wie bereits erwähnt, mit dem sogenannten Designer unter Verwendung der Windows Forms Schnittstelle erstellt worden und besteht neben dem Fenster für die MIDI-Optionen (siehe unten) aus dem in Abbildung 7.3 dargestellten Hauptfenster. Daher erinnert die GUI auch stark an das klassische Windows Design.

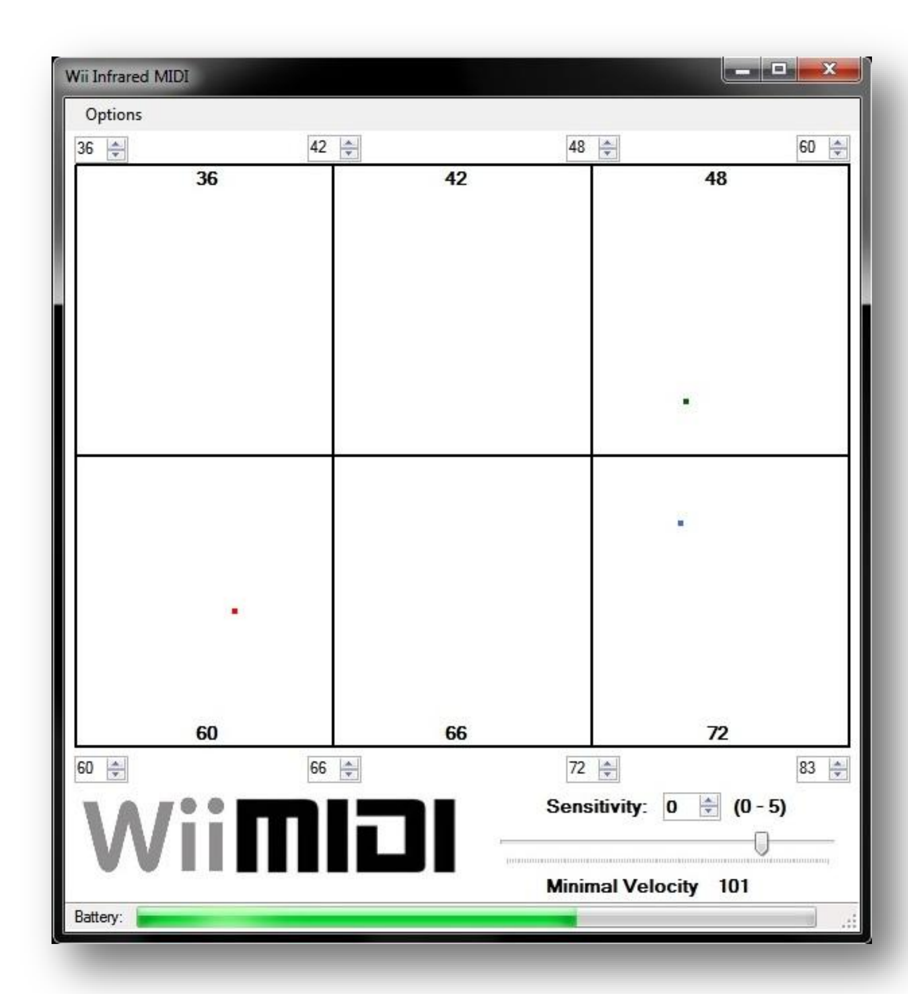


Abb. 7.3: Wii Infrared MIDI GUI

Das wohl wichtigste und dem daher auch am meisten Platz zugeordnete Elemente ist die in sechs Rechtecke unterteilte Arbeitsfläche, mit der die Positionen und Bewegungen der Infrarot-Signale wiedergegeben werden. Diese wird im Designer zunächst als derartige *PictureBox* angelegt. Für die verschiedenen IR-Quellen sind die drei Farben Rot, Blau und Grün verwendet worden.

Wird die Auslösebewegung identifiziert und dadurch ein MIDI-Befehl gesendet, so leuchtet das betätigte Feld kurz in grüner Farbe auf, um dem Benutzer visuelles Feedback zu geben.

Daneben ist jedes der Felder mit einer MIDI-Notenwertanzeige ausgestattet, sodass diese immer direkt abgelesen werden können. Über bzw. unter den im weiteren als Pads bezeichneten Rechtecke sind, jeweils davor und dahinter, sogenannte *NumericUpDown*-Schaltelemente integriert, die den Bereich bestimmen in dem

der Notenwert des Feldes durch die Switch-Befehle variiert werden kann, damit es zu keinen Überschneidungen zwischen den einzelnen Abschnitten kommt. Feld D, unten links in Abbildung 7.3, kann beispielsweise die MIDI-Notenwerte zwischen 60 und 66 annehmen.

Desweiteren wird ein weiterer *NumericUpDown* verwendet, um die *Sensitivity*, also die Empfindlichkeit des Systems auf die Bewegung der IR-Punkte zu regeln. Darunter ist eine sogenannte *Trackbar* angebracht mit dessen Hilfe die Mindest-Velocity des ausgelösten MIDI-Befehls festgelegt wird und die vor allem dazu dient, die unterschiedliche Lautstärke beim Aktivieren des Samples anzupassen, falls die Velocity sehr differierende Werte annimmt. Der Schriftzug *WiiMIDI* besitzt keine Funktion und fungiert allein als grafische Aufwertung der Softwareoberfläche. Am unteren Ende dieser ist die Anzeige der restlichen Batterieleitung in einer grünen *ProgressBar* visualisiert. Daran ist auch immer zu erkennen ob eine Wiimote dem System angeschlossen ist. Falls dies nicht der Fall ist, ist die gesamte Batterieleiste leer.

Eine weitere wichtige Komponente ist das eingebaute Optionsmenü in der oberen Leiste des GUI, mit dem sich die MIDI-Optionen anwählen lassen (siehe Abb. 7.4). In Visual Basic bedeutet ein neues Fenster immer auch ein komplett neues Windows Forms Element mit einer eigenen Klasse. Dieses Element ist hierbei jedoch mit seinem ganzen visuellen Aufbau und allen seinen in der Klasse *MidiOptions* implementierten Methoden komplett aus der Wii-Drum-Synth Anwendung übernommen. Es können damit *„Output Device“* MIDI-Programm (Channel) und MIDI-Instrument gewählt werden. Letztere beiden sind aber für den eigentlichen Gebrauch irrelevant. In der Einstellung den Punkt *„Output Device“* betreffende, kann der MIDI-Treiber bzw. das Verbindungselement ausgewählt werden, das letztlich die MIDI-Befehle dem Softwareinstrument weiterleitet. Für Windows PCs ist hierbei grundsätzlich der dem Computer eingebaute GS Microsoft Wavetable Synthesizer ausgewählt, der prinzipiell auch als Klangerzeuger dienen kann. Sind jedoch andere MIDI-Treiber angeschlossen, können diese gewählt werden.

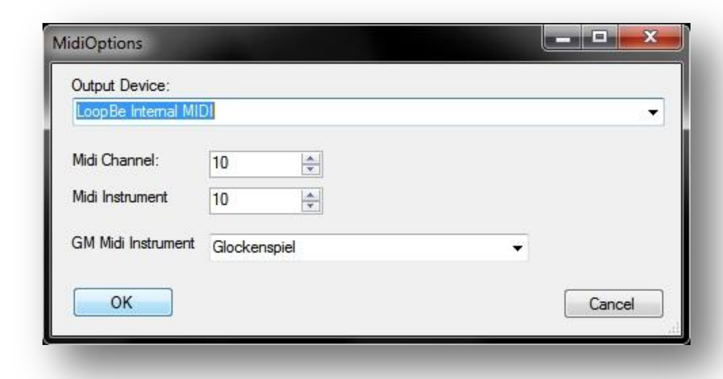


Abb. 7.4: Wii Infrared MIDI Optionsmenü

Im weiteren Verlauf wird nun explizit auf die Umsetzung der Funktionen in der Implementierung eingegangen.

7.3.2. Softwareimplementierung

Bevor nun die einzelnen Methoden detailliert vorgestellt werden, wird noch kurz auf wichtige in Wii Infrared MIDI eingesetzte C# Syntax eingegangen und ein kleiner Überblick über das Gesamtprogramm gegeben, um den Sinn der einzelnen Methoden später im Gesamtkontext richtig einordnen zu können. Es muss an dieser Stelle erwähnt werden, dass in diesem Kapitel nur Ausschnitte aus der Codestruktur gezeigt werden um Funktionen darzulegen. Der gesamte Code ist in ausführlich kommentierter Form in Anhang B einzusehen.

7.3.2.1. C# Syntax in Wii Infrared MIDI

Zu der sehr allgemein gehaltenen Einführung in Visual Basic in Kapitel 7.1 kommen nun noch einige notwendige die Programmiersprache betreffende Erklärungen, die das Verständnis zur Ausführung der Methoden hoffentlich fördert.

Vor der Erstellung des Quelltexts müssen zunächst die Verweise zu den verwendeten ***Namespaces*** hergestellt werden, zu denen auch die Managed Libraries zählen. Dafür wird neben der Einbeziehung in den Projektmappe von Visual Basic auch noch der ***using***-Befehl benötigt, der unmittelbar am Anfang des Codes steht (siehe Anhang B, Seite 1).

Zunächst wird die Struktur der verschiedenen Klassen betrachtet. Die eigens entwickelte Programmstruktur mit ihren einzelnen Methoden ist der Hauptklasse *MainForm* untergeordnet. Neben dieser muss die für das MIDI-Optionsmenü zuständige *MidiOptions*-Klasse erwähnt werden, die zwar dem Quelltext angehängt, ihr aber weiterhin keine gesonderte Erklärung beigefügt wird, da sie lediglich aus dem Wii-Drum-Synth-Programm übernommen wurde. Klassen aus der WiimoteLib und dem C# MIDI Toolkit werden dagegen nur instanziert, denn auf die Objekte oder Klassen der Managed Libraries kann nur zugegriffen werden, wenn zu diesen zuvor eben eine Instanz gebildet wurde. Das betrifft das essentielle ***Wiimote***-Objekt aus der WiimoteLib und die ***Synth***-Klasse aus dem MIDI Toolkit (siehe Anhang B, Seite 1).

Daraufhin folgt die Definition globaler Variablen, die im Gegensatz zu lokalen Variablen, welche nur innerhalb einer Methode definiert werden können, in der gesamten Klasse gelten. Dabei handelt es sich um Variablen, die unter anderem als Grundlage der Berechnung der Bewegung der Infrarotpunkte dienen, gewisse Schwellenwerte bilden und damit Grenze von Auslösungsvorgängen sind, Notenwerte und deren textliche Entsprechung darstellen oder eine Grundlage der Positionsanzeige auf der GUI beschreiben (siehe Anhang B, Seite 2-3).

Lokale Variablen hingegen können von einer zur anderen Methode übergeben werden. In diesem Fall steht die übergebene Variable unmittelbar hinter der Methodendeklaration in Klammern. Allen Variablen ist immer ein Name, ein Datentyp und dessen Wert zugeordnet, der manchmal zu Beginn bereits definiert ist, oft aber auch im Laufe der Anwendung erst zugeordnet wird. Für Wii Infrared MIDI werden vor allem folgende Datentypen verwendet, die mitunter auch in ineinander konvertiert werden:

- *Integer* (Kurzform *int*): ganze Zahl
- *float*: Gleitkommazahl
- *decimal*: Festkommazahl
- *string*: Zeichenkette
- *bool*: Schaltvariable (nur *true* oder *false*)

Alle Klassen, Methoden und Variablen können zusätzlich noch über ihnen vorangestellte derartige Zugriffsmodifizierer wie *internal, private, public* und *protected* verfügen, die darüber entscheiden in welcher Struktur über Klassen hinaus auf das jeweilige Ziel zugegriffen werden kann. Für den Zugriff innerhalb einer Klasse wie in *MainForm* ist dies jedoch irrelevant. Zu weiteren Informationen bezüglich der Einbindung von Wii Infrared MIDI in andere Programmstrukturen, wird auf KÜHNEL (2010) hingewiesen.

Die eigentlichen Operationen erfolgen aber innerhalb der Methoden. Dazu gehören neben mathematischen und logischen Berechnungsmethoden unter anderem auch sogenannte Schleifen die Befehle ausführen, falls oder solange eine gewisse Bedingung erfüllt ist. Eine dieser Schleifen ist die *if*-*else*-Anweisung, die hier sehr oft verwendet wird und daher kurz einer Erläuterung bedarf. Untenstehendes Beispiel zeigt dessen klassische Syntax.

```
if (Bedingung)
Anweisung 1;
else
Anweisung 2;
```

Nach dem *if* -Befehl steht in Klammern die Bedingung, die zum Ausführen der Anweisung 1 erfüllt sein muss. Ist das nicht der Fall wird Anweisung 2 ausgeführt. Es besteht allerdings auch die Möglichkeit auf den *else*-Teil zu verzichten, falls nur eine Anweisung bei einer bestimmten Bedingung interessiert. Weitere Operationen werden nicht besonders erklärt, es wird auf GUNNERSON (2000) verwiesen.

Daneben soll vor der Beschreibung der Implementierung noch kurz auf die programmtechnische Definition eines Events (zu Deutsch: Ereignis) eingegangen werden. Der Methodenaufruf in C# erfolgt in diesem Programm zum Einen wenn eine vorig ausgeführte Methode diesen Aufruf befiehlt oder im Falle eines Events, wenn ein bestimmtes Ereignis ausgelöst wird, dass wiederum die Methode aufruft. Im vorliegenden Fall ist dieses Ereignis zum Beispiel, dass ein bestimmter Datenparameter, der von der Wiimote gesendet wird, verändert wird, wodurch dann der Methodenaufruf erfolgt.

Ein Delegat wiederum beschreibt nur die Weiterleitung des Methodenaufrufs des Ereignisses an die eigentliche Methode. Diese Vermittlung muss jedoch manchmal eben durch das Delegat angepasst werden, um die Funktion des Gesamtsystems aufrechtzuhalten (vgl. KÜHNEL 2010).

Zur letzten wichtige Syntax-Form, die hier kurz erklärt werden, zählt der sogenannte Timer. Dieser entspricht einer Methode, die in einem gewissen, definierten Zeitabstand immer wieder aufgerufen und ausgeführt wird und dessen Intervall in sogenannten Ticks festgelegt wird. Ein Tick entspricht dabei einer Millisekunde.

7.3.2.2. Koordinatensystem

An dieser Stelle soll noch eine Anmerkung zur Anlage des Koordinatensystems gemacht werden, das sich als recht komplex herausstellt.

Bei ihrer eigentlichen Verwendung wird die Wiimote häufig als Zeiger verwendet, die eine feste über oder unter dem Fernsehgerät installierte Infrarot-LED-Leiste anvisiert um einen Cursor im Videospiel zu steuern. Wird nun die Wii-Remote beispielsweise nach links bewegt um den Zeiger in diese Richtung zu lenken, bewegen sich aus Sicht der IR-Kamera die LEDs nach rechts, wird sie nach unten bewegt, so wandern die IR-Punkte aus Kamera-Sicht nach oben. Um dies zu kompensieren ist der Wiimote im Bezug auf die Erkennung von Infrarotsignalen sozusagen direkt ein in x-Richtung gespiegeltes Koordinatensystem eingesetzt worden. Abbildung 7.5 zeigt dieses Bezugssystem in der Frontansicht der Kamera.



Abb. 7.5: Infrarotkamera-Bezugssystem



Abb. 7.6: gedrehtes Bezugssystem

Jegliche horizontale Bewegung von IR-Punkten würde die Wiimote seitenverkehrt aufnehmen. Um dem ganzen entgegenzuwirken kann eine programmtechnische Umrechnung der Koordinaten erfolgen oder wie bei der Wii-Infrared-MIDI-Anwendung die Wiimote einfach auf der Knopfleiste platziert werden (siehe Abbildung 7.6). Insgesamt werden dadurch die IR-Punkte ihrer realen Bewegung entsprechend in der Software abgebildet, es muss jedoch bedacht werden, dass die positive y-Achse in die Richtung vertikal nach unten zeigt und damit beispielsweise eine IR-Bewegung nach unten eine positive Geschwindigkeit bezüglich der y-Achse aufweist. Nach diesen Erläuterungen folgt nun zunächst eine Zusammenfassung der verschiedenen Methoden.

7.3.2.3. Übersicht der Methoden

Wie vielleicht bis jetzt in der Ausführung zur Softwareimplementierung schon zu erkennen ist, bilden die Methoden, in denen im Endeffekt alle Operationen ausgeführt werden, das Herzstück der Anwendung. Daher erscheint es sinnvoll zunächst kurz ein Resümee dieser Methoden und ihrer Aufgaben der eigentlichen detaillierten Ausführung voranzustellen. Tabelle 4.1 zeigt alle benutzten Methoden der *MainForm* -Klasse und, dem übergeordnet, eine Einteilung in ihre jeweiligen Aufgabenbereiche.

Tabelle 4.1: Übersicht der einzelnen Methoden

Überbegriff	Methode
Start-Methoden	MainForm()
	Form1Load()
Berechnungsmethoden	berechne_V1_y()
	berechne_V2_y()
	berechne_V3_y()
	berechne_A1_y()
	berechne_A2_y()
	berechne_A3_y()
	berechne_V1_x()
	berechne_V2_x()
	berechne_V3_x()
	berechne_X_Gap12()
	berechne_X_Gap23()
	berechne_X_Gap13()
Switch-Forward Methoden	A_Switch_fwd()
	B_Switch_fwd()
	C_Switch_fwd()
	D_Switch_fwd()
	E_Switch_fwd()
	F_Switch_fwd()
Switch-Back Methoden	A_Switch_back()
	B_Switch_back()
	C_Switch_back()
	D_Switch_back()
	E_Switch_back()
	F_Switch_back()
Wiimote-Gestenerkennungsmethoden	wm_WiimoteChanged(object sender, WiimoteChangedEventArgs e)
	wm_WiimoteChanged_Switch(object sender, WiimoteChangedEventArgs e)
Wiimote GUI-Zeichenmethoden	UpdateState(WiimoteChangedEventArgs args)
	UpdateWiimoteChanged(WiimoteChangedEventArgs args)
	UpdateIR(IRSensor irSensor, Color color)
Timer-Methoden	timer1_Tick(object sender, EventArgs e)
	timer2_Tick(object sender, EventArgs e)
	timer3_Tick(object sender, EventArgs e)
	timer4_Tick(object sender, EventArgs e)
	timer5_Tick(object sender, EventArgs e)
GUI Methoden	mIDISetupToolStripMenuItem_Click(object sender, EventArgs e)
	trackBar1_Scroll(object sender, EventArgs e)
	numericUpDown1_ValueChanged(object sender, EventArgs e)
	numericUpDown2_ValueChanged(object sender, EventArgs e)
	numericUpDown3_ValueChanged(object sender, EventArgs e)
	numericUpDown4_ValueChanged(object sender, EventArgs e)
	numericUpDown5_ValueChanged(object sender, EventArgs e)
	numericUpDown6_ValueChanged(object sender, EventArgs e)
	numericUpDown7_ValueChanged(object sender, EventArgs e)
	numericUpDown8_ValueChanged(object sender, EventArgs e)
	numericUpDown9_ValueChanged(object sender, EventArgs e)
	Form1_FormClosing(Object sender, FormClosingEventArgs e)

7.3.2.4. Beschreibung der Methoden

An dieser Stelle muss zunächst darauf hingewiesen, dass einige Methoden auf dem Programm „WiimoteTest“ von PEEK (2009) und der entwickelten „Wii Drum Synth“ Applikation basieren (vgl. WIIDRUMSYNTH.CODEPLEX), deren beider Quelltext auf CODEPLEX.COM öffentlich und frei verfügbar ist. Die Beschreibung der Methoden erfolgt nun der Reihenfolge der Liste 4.1 und ihrer Einteilung nach. Falls es zur besseren Übersicht dient, werden die dazugehörigen Befehle und Operationen, die darin jeweils implementiert sind, gegebenenfalls in stichpunktartiger Form beschrieben. Dabei wird immer auf den zugehörigen Quelltext-Abschnitt hingewiesen.

7.3.2.4.1. Start-Methoden

Im Anschluss an die nötige Definition der globalen Variablen, die in Kapitel 7.3.2.1 bereits behandelt wurde, werden nun die Start-Methoden erläutert, die auch chronologisch gesehen als Erstes ausgeführt werden.

`public MainForm()`

Dazu gehört auch die bei Programmstart direkt aufgerufene Methode *MainForm*. Hierin werden folgende Arbeitsschritte erledigt (siehe Anhang B, Seite 3-4):

- Erforderliche Methode für die Designerunterstützung *InitializeComponent* aufrufen
- Objekt *synth* aus der Klasse *Synth* definieren, das später die MIDI-Befehle sendet
- Das *Bitmap* *b* als Zeichenoberfläche *g* der *Graphics* -Klasse definieren. *b* sowie *g* wurden zuvor neben der Definition der globalen Variablen mittels der Instanzierung einer Graphics- und einer *Bitmap*-Klasse definiert
- Um den Mindest-Velocitywert auf die GUI zu projizieren wird die *string*-Variable *Text_Vel_min* als Entsprechung der *int*-Variable *Velocity_min* definiert, die danach als Schriftzug *Label_Vel_min.Text* auf die Bedienoberfläche übertragen wird
- Sicherstellen, dass die Notengrenzen-Schaltelemente *numericUpDown6* und *numericUpDown5* denselben Wert besitzen

`private void Form1_Load(object sender, EventArgs e)`

Die zweite der Startmethoden wird beim Öffnen des Hauptfensters aufgerufen. Hier werden die meisten der Methoden des Wiimote-Objekts aufgerufen und damit die Einstellungen der Wii-Remote angepasst (siehe Anhang B, Seite 4-5):

- Erstellen zweier WiimoteChanged-Delegaten `EventHandler aus der WiimoteChangedEventArgs`-Klasse des Wiimote Objekts zu den Methoden `wm_WiimoteChanged_Switch` und `wm_WiimoteChanged`, die somit immer durch das Ereignis eines sich ändernden Datenstatus der Wiimote aufgerufen werden. Die Übergebenen Parameter `e` entsprechen den aktuellen Wiimote-Statusdaten
- Verbinden der Wiimote mit dem Aufrufen der `wm.Connect` Methode des Wiimote-Objekts
- Über `wm.SetLEDs(false, false, true, true)` die LEDs 3 und 4 der Wiimote aktivieren (dient zur Erkennung der Verbindung der Wiimote mit dem Programm)
- Datenreporttyp und Sensivität der Wiimote über `wm.SetReportType(InputReport.IRAccel, IRSensitivity.Maximum, true)` festlegen
- Datenreporttyp der Infrarotkamera mit `wm.WiimoteState.IRState.Mode = IRMode.Extended` festlegen
- Batteriestatus als Variable `float B` von der Wiimote abrufen, diese anschließend in die Variable `int Bat` überführen, um damit danach den Wert der Batterieleiste zu definieren
- Erstellen der Timer-Methoden durch Definition ihrer Delegaten, Aktivieren der Timer und Festlegen ihrer Intervalle durch die globale Variable `all_timer_interval = 200` (Timer 1-3) und die festen Werte `800` (Timer 4) und *50* (Timer 5)

7.3.2.4.2. Berechnungsmethoden

Wie in Kapitel 5.1 zur Konzeption bereits vorgegriffen, soll die Gestenerkennung zur Aktivierung der Samples durch die Schlagbewegung über die Beschleunigung der IR-Punkte in negative y-Richtung erfolgen.

Da die Wiimote allerdings nur Positionsdaten der Infrarotbewegungen zur Verfügung stellt, müssen diese durch Differentiation zunächst in die Geschwindigkeits- und anschließend durch erneutes Ableiten in die Beschleunigungswerte transformiert werden. Dabei erfolgen die Berechnungen für beide Koordinatenrichtung getrennt und dies jeweils für die drei verschiedenen Infrarotsignale. In allen Fällen wird die Differentiation durch Bildung eines Differenzenquotienten (siehe unten) angenähert, der neben dem Zeitintervall immer aus dem aktuellen und dem jeweilig davor von der Wiimote gesendeten Positionswert besteht. Die Zeitspanne wird hier durch die globale Variable `float t = 10` beschrieben, da der Zeitabstand zwischen zwei eingehenden Positionsdatensignalen, die von der Wiimote mit etwa 100Hz gesendet werden, nach eigenen Messungen relativ genau 10 Millisekunden beträgt. Als letzter Schritt wird die globale Variable, die den vorigen Positionswert beschreibt mit dem aktuellen Wert überschrieben.

private void berechne_V1_y()

Das folgende Beispiel zeigt die Methode zur Berechnung der Geschwindigkeit des ersten Infrarotpunktes in y-Richtung:

```
private void berechne_V1_y()
{
        float Y1 = wm.WiimoteState.IRState.IRSensors[0].RawPosition.Y;
        V1_y = (Y1 - Y1_vorher) / t;
        Y1_vorher = Y1;
}
```

Zunächst wird der Momentanwert der Position über das Wiimote-Objekt abgerufen und als *float*-Variable mit entsprechendem Namen definiert, anschließend folgt die Bildung des Differenzenquotients und abschließend die Überspeicherung des alten Positionswerts. Alle anderen Geschwindigkeits-Berechnungsmethoden *berechne_V2_y, berechne_V3_y, berechne_V1_x, berechne_V2_x, berechne_V3_x* sind nach dem genau gleichen Prinzip aufgebaut und unterscheiden sich lediglich hinsichtlich des verwendeten Positionswerts.

private void berechne_A1_y()

Auch die Berechnung der Beschleunigungswerte, für die die Methode *berechne_A1_y* als Exempel dienen soll, lehnt sich an diese Systematik an.

```
private void berechne_A1_y()
{
        berechne_V1_y();

        A1_y = (((V1_y - V1_y_vorher) / t) +
               (((V1_y - V1_y_vorher) / t) * (int)Sens * 1 / 5)) * 50;

        if (V1_y < 0)
            A1_y = 0;

        if (A1_y < 0)
            A1_y = 0;

        V1_y_vorher = V1_y;
}
```

Zu Beginn wird zur Berechnung des momentanen Werts der Geschwindigkeit des IR-Punktes in die jeweilige Richtung die passende Methode aufgerufen. Danach erfolgt, analog zur Geschwindigkeitsberechnung, die Aufstellung des Differenzenquotients. Hierzu kommt aber noch eine Addition eines Werts, die den Sensitivity-Faktor, der auf der GUI durch einen horizontalen Regler einstellbar ist (siehe Kapitel 7.3.2.4.7), berücksichtigt. Der Wert der schließlich addiert wird, entspricht einem Fünftel der eigentlichen Beschleunigung multipliziert mit dem benannten Faktor. Dadurch soll der Beschleunigungswert angepasst werden können, falls der Anwender die IR-Bewegungen in verschiedenen Entfernungen von der Wiimote und somit der IR-Kamera ausführt, um auch in diesem Fall die Überschreitung des festgelegten Schwellenwerts

zur Auslösung der Samples (siehe Kapitel 7.3.2.4.4) zu gewährleisten. Eine Alternative dazu wäre natürlich die Variation des Schwellenwerts anhängig vom Sensitivity-Faktor gewesen. Eine Kalibrierung des Systems auf einen festen Mindestwert hat sich jedoch als günstiger erwiesen. Um die Betätigung der Samples durch vertikal nach oben gerichtete Bewegungen auszuschließen wird der Beschleunigungswert, falls dieser selbst oder die Geschwindigkeitskomponente der IR-Punkts negativ sind (siehe Kapitel 7.3.2.2 zum Koordinatensystem), gleich null gesetzt. Als letzter Programmschritt wird hier, entsprechend den Geschwindigkeitsberechnungen, der alte Wert der Geschwindigkeit mit dem neuen überschrieben, um die Berechnungen über den Differenzenquotient zu ermöglichen. Die Methoden *berechne_A2_y* und *berechne_A3_y* sind gleich aufgebaut.

`private void berechne_X_Gap12()`

Als letzte der Berechnungsmethoden folgt hier die mathematische Formulierung zur Abstandsberechnung zwischen den einzelnen x-Koordinaten der Infrarotsignale (siehe Anhang B, Seite 6). Hier werden zunächst wieder die erforderlichen Positionsdaten über das *wiimote*-Objekt als *float*-Variable abgerufen, anschließend die Differenz der Werte und schließlich noch der Betrag gebildet, da das Vorzeichen der Differenz bei derlei Abstandsberechnungen nicht relevant ist. *berechne_X_Gap23* und *berechne_X_Gap13* beinhalten abgesehen von den differierenden IR-Punkten die gleiche Codestruktur.

7.3.2.4.3. Switch-Methoden

Anschließend werden die Methoden zur Ausführung der Switch-Back- sowie Switch-Forward-Befehle der verschiedenen Felder gezeigt. Diese dürfen aber auf keinen Fall mit den Methoden der Gestenerkennung zur Auslösung der Switch-Methoden verwechselt werden, die im Abschnitt Wiimote-Methoden dargelegt werden. Alle sechs Felder sind mit Großbuchstaben entsprechend Abbildung 7.7 benannt worden. Nach diesen Buchstaben werden ebenfalls die *int*-Variablen der MIDI-Notenwerte bezeichnet. Stellvertretend für die anderen Switch-Forward-Methoden *B_Switch_fwd, C_Switch_fwd, D_Switch_fwd, E_Switch_fwd, F_Switch_fwd* wird nun *A_Switch_fwd* erklärt.

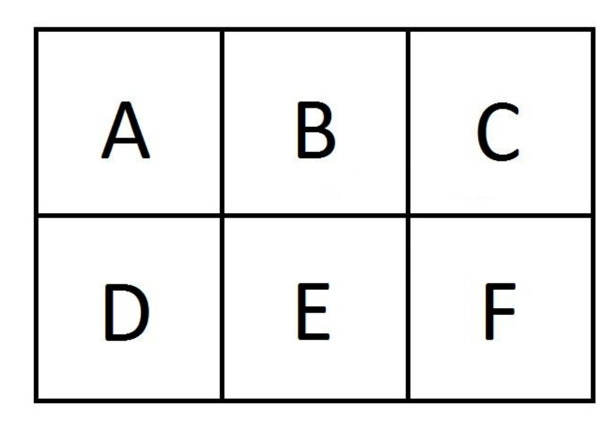


Abb. 7.7: Benennung der Felder

private void A_Switch_fwd()

Die Hauptaufgabe besteht hierbei in der einfachen Addition des Werts 1 zum entsprechenden MIDI-Notenwert des Feldes. Zusätzlich soll sichergestellt werden, dass dieser Notenwert, den durch die *NumericUpDown*-Schaltelemente begrenzenden Notenbereich (siehe Kapitel 7.3.1 zur Bedienoberfläche) des Feldes nicht überschreitet. Dazu wird die Notenwertaddition, falls die gegebene Grenze erreicht ist, einfach wieder revidiert (siehe Anhang B, Seite 7).

private void A_Switch_back()

In gleicher Weise spielen sich die Switch-Back-Befehle ab. Zunächst wird der dem Feld entsprechende Notenwert um 1 verringert, bevor überprüft wird ob diese untere Grenze, ebenfalls durch ein Schaltelement festgelegt, unterschritten ist. Sollte dies der Fall sein, so wird die Notenwertsubtraktion wieder aufgehoben (siehe Anhang B, Seite 8). Die fünf anderen Switch-Back-Methoden entsprechen dieser bis auf die Verwendung des jeweiligen Notenwerts.

7.3.2.4.4. Wiimote-Gestenerkennungsmethoden

Der Mittelpunkt der Anwendung besteht aus den Wiimote-Methoden, in denen die Gestenerkennung abläuft, die letztlich zum Auslösen der gewünschten Befehle durch eine bestimmte Eingabe über den Nutzer führt. Als erstes wird die Methode angeführt, die für die Auslösung der Samples zuständig ist.

void wm_WiimoteChanged(object sender, WiimoteChangedEventArgs e)

Durch den Eventhandler (siehe *Form1_Load*-Methode) wird diese Methode immer aufgerufen sobald sich die Statusinformationen der Wiimote ändern. Alle nötigen Wiimote-Statusdaten werden durch den Parameter *e* weitergegeben. Zur Vorbereitung der Erkennung der Auslösungsbewegung laufen daher folgende Arbeitsschritte ab:

- Abrufen der x- und y-Positionsdaten der drei Infrarotsignale in *float*-Variablen über den Parameter *e*
- Deaktivieren des vierten Infrarotsignals, das nicht benötigt wird und ansonsten nur zu programmtechnischen Missverständnissen führt
- Aufrufen der Methoden zur Beschleunigungsberechnung in y-Richtung
- *UpdateState(e)*- Methode aufrufen unter Übergabe der Parameter *e*

Anschließend wird der eigentliche Teil der Methode ausgeführt. Zunächst wird geprüft ob die Beschleunigungswerte die vorgegebenen Bedingungen erfüllen, also ob die IR-Bewegung etwa einer

Schlagbewegung entspricht. Neben einem Schwellenwert zur Mindestbeschleunigung *A_y_Thres*, der überschritten werden muss, ist zusätzlich noch eine Grenze zu Maximalbeschleunigungen *A_y_Max* eingebaut, die verhindern soll, dass beispielsweise der Eintritt eines IR-Signals in den von der Kamera wahrgenommenen Raum, bei dem sehr hohe Beschleunigungswerte auftreten können, ein Auslösen eines Samples nach sich zieht. Diese Werte sind experimentell entwickelt worden.

Sind die Bedingungen erfüllt, wird als nächster Schritt der MIDI-Parameter der Velocity in einer *float* *veloc*-Variable definiert, die dem Beschleunigungswert minus dem erforderlichen Schwellenwert, skaliert auf die vom MIDI-Standard vorgegebene Velocity-Skala der Werte von 1 bis 127, entspricht. Es folgen eine Konvertierung der Velocity-Variable in den Typ *int*, der zur Verarbeitung notwendig ist, eine Überprüfung ob der Velocity-Wert auch über dem auf der GUI einstellbaren Mindest-Velocity liegt, die gegebenenfalls eine Anpassung nach sich zieht, und ein Sicherstellen, dass der Velocity-Wert das Maximum von 127 der eben erwähnten MIDI-Velocity-Skala nicht überschreitet.

Hieran erfolgt, durch die Positionsdaten der Wiimote, die Zuordnung der Auslösungsbewegung zu einem der sechs Felder mittels geschachtelter *if*-Schleifen, die bestimmen welcher der Notenparameter A bis F ausgewählt wird.

Jetzt erst wird der MIDI-Befehl mit Hilfe des zu Anfang erstellten *synth*-Objekts durch den Befehl *PlayNote(note, veloc)* gesendet. In der *note*-Variablen ist dabei die Notenwertinformation eines der sechs Felder gespeichert und der *veloc*-Parameter bestimmt die ausgehende Velocity. Die Länge der MIDI-Note kann allerdings nicht beeinflusst werden. Es wird immer nur ein kurzer MIDI-Befehl gesendet, der das Gesamtsystem wie in Kapitel 5 schon angedeutet wird, eher für Samples qualifiziert, die auch dabei komplett abgespielt werden, im Gegensatz zu virtuellen Instrumente wie etwa Streicher oder Klaviere bei denen es ebenfalls auf das Halten einer oder mehrerer Noten ankommt.

Durch die verwendete Methode der Gestenerkennung tritt jedoch der Fehler auf, dass das gewünschte Sample gleich diverse Male hintereinander ausgelöst wird, da die Wiimote bei einer Datentransferfrequenz von 100Hz sehr viele Statusinformationen sendet und darum der Schwellenwert zur Mindestbeschleunigung gleich mehrere Male überschritten wird.

Um dieses Problem zu lösen ist die Funktion eingebaut, dass, sobald ein MIDI-Befehl durch das *synth*-Objekt gesendet wird, eine Timer-Variable *peak_filter* vom Typ *bool* auf *false* gesetzt wird, die von dem Timer 4 in einem dem Intervall von 200 Millisekunden immer wieder auf true gesetzt wird. Durch eine dem *synth*-Befehl vorgeschaltete *if*-Schleife wird dieser nur betätigt, falls die Timer-Variable auch den Wert *true* besitzt.

Zusammengefasst bedeutet dies, dass nach dem Auslösen eines MIDI-Befehls mit einem der drei Infrarotsignale für 0,2 Sekunden kein weiterer MIDI-Befehl mehr mit demselben Signal ausgelöst werden kann. Somit wird wie gewünscht bei jeder Schlagbewegung auch nur einmalig ein Sample betätigt. Dieser Prozess läuft allerdings bei alle drei IR-Signalen separat ab und demnach sind dafür auch die drei unterschiedlichen Timer 1,2 und 3 erstellt worden.

Um das in Kapitel 7.3.1 beschriebene visuelle Feedback durch kurzzeitiges Einfärben des betätigten Feldes nach erfolgreicher Sample-Auslösung zu generieren, wird die Variable *feedback* auf *true* gesetzt. Solange ihr dieser Wert zugeordnet ist, wird das jeweilige Feld durch ein einen Befehl in *UpdateWiimoteChanged* in grüner Farbe dargestellt. Da jedoch nur ein kurzzeitiges aufleuchten des Bereichs erwünscht ist, stellt der fünfte Timer zu Beginn seines relativ kurzen Zeitintervalls sicher, dass der *feedback* Variable wieder der Wert *false* zugeordnet wird, damit das Aufleuchten des Feldes beendet wird.

Der Hauptteil der Methode zur Gestenerkennung ist dabei insgesamt drei Mal, für jedes der drei IR-Signale separat in der Syntax niedergeschrieben (siehe Anhang B, Seite 9-11)

`void wm_WiimoteChanged_Switch(object sender, WiimoteChangedEventArgs e)`

Im Unterschied zur soeben dargelegten Methode, dient *wm_WiimoteChanged* nur zur Erkennung der Eingabe, nicht aber zur Ausführung des Befehls. Dafür wird auf die Switch-Methoden in Kapitel 7.3.2.4.3 verwiesen. Die Similarität zur letzten Methode besteht jedoch in der Erkennungsweise der Geste. Durch den Eventhandler führen sich ändernde Wiimote-Statusdaten, die dann über den Parameter *e* übergeben werden, auch hier immer zum Methodenaufruf. Zur Vorbereitung auf den Entscheidungsprozess der Methode werden untenstehende Vorgänge analog zu *wm_WiimoteChanged* ausgeführt:

- Abrufen der x- und y-Positionsdaten der drei Infrarotsignale in *float*-Variablen über den Parameter *e*
- Deaktivieren des vierten Infrarotsignals
- Aufrufen der Methoden zur Geschwindigkeitsberechnung in x-Richtung
- Aufrufen der Methoden zur Berechnung der Positionsdifferenzen in x-Richtung

Für die genaue Erklärung des Switch-Befehls wird an dieser Stelle zunächst noch einmal auf Abbildung 5.5 in Kapitel 5.1 auf Seite 41 hingewiesen, die den Prozess anschaulich visualisiert. Die Anforderungen durch die *if* -Schleife zur Auslösung des Befehls bestehen in vier einzelnen Bedingungen.

Zunächst müssen die Geschwindigkeiten der beiden IR-Punkte, die die horizontal dynamische Auslösebewegung beschreiben, einer geforderten Mindestgeschwindigkeit entsprechen. Für die Switch-Forward-Auslösung muss hierbei einfach der Schwellenwert *V_x_Thres_plus* übertroffen werden, bei der Betätigung der Switch-Back-Befehle kehren sich jedoch die Vorzeichen um, denn eine vom Anwender dafür ausgeführte Auslösebewegung nach links zieht eine negative x-Geschwindigkeitskomponente nach sich. Damit folgt, dass hier ein gewisser Schwellenwert *V_x_Thres_minus* unter- statt überschritten werden muss. Beide Anforderungswerte besitzen denselben Betrag, unterscheiden sich logischerweise jedoch hinsichtlich des Vorzeichens.

Zur Verhinderung von zufällig betätigten Auslösung einer Methode bei Eintreten der LEDs in die Kamerafläche ist auch bei den Switch-Befehlen, wie schon bei der Sampleauslösung, eine positive

Maximalgrenze der Geschwindigkeit *V_x_Max* für die IR-Bewegungen beim Forward-Befehl und eine entsprechende negative Maximalgrenze *V_x_Min* für den Back-Befehl eingebaut. Diese Grenzen besitzen ebenfalls den gleichen Betrag.

Desweiteren soll zugleich sichergestellt werden, dass sich die IR-Punkte bei der Betätigungsbewegung untereinander befindet. Dafür werden die Berechnungen zu den Positionsdifferenzen bezüglich der x-Achse benötigt. Die dritte Bedingung stellt demnach sicher, dass diese Differenz einen Maximalwert des Positionsunterschieds *X_Gap_Max* nicht überschreitet. Auf die Vorzeichenproblematik muss hierbei nicht geachtet werden, da die berechneten x-Koordinatendifferenzen als Betrag vorliegen (siehe Kapitel 7.3.2.4.2 zu Berechnungsmethoden).

Sind nun alle Bedingungen erfüllt wird über die Position des Infrarotpunktes, der die Auswahlbewegung beschreibt, die Methode ausgesucht, die angewählt wird und letztlich den Switch-Befehl im richtigen Feld ausführt. Zur Vermeidung einer Mehrfachauslösung wird dabei, analog zur Methode der Sampleauslösung, eine Timervariable *peak_filter_X* auf den Wert *false* gesetzt. Im Gegensatz zur Methode zur Sampleauslösung wird aber nur eine Timervariable und daher nur ein Timer für alle Switch-Befehle benötigt, da immer nur maximal einer der Switch-Befehle gleichzeitig ausgeführt werden kann. Außerdem ist das Timerintervall mit 800 Millisekunden auch etwas großzügiger bemessen, da die Switch-Bewegung nicht so schnell nacheinander ausgeführt werden kann. Erst nach diesen 0,8 Sekunden wird die Timervariable, die durch eine *if* -Schleife noch vor allen anderen Bedingungen überprüft wird, wieder auf *true* gesetzt und es kann eine erneute Eingabe des Switch-Befehls erfolgen (siehe Anhang B, Seite 11-17).

7.3.2.4.5. Wiimote-GUI-Zeichenmethoden

In den drei derartigen Zeichenmethoden, die ihre Grundlage ebenfalls auf dem *wiimote*-Objekt bilden, geht es um die Visualisierung der Bewegung der Infrarotsignale auf der GUI und dessen Umfeld-Gestaltung.

```
public void UpdateState(WiimoteChangedEventArgs args)
```

Wie bereits erwähnt wird `UpdateState` durch *wm_WiimoteChanged* und somit auch durch jede Statusdatenänderung der Wiimote aufgerufen. Die gesamte Methode besitzt nur eine einzige Aufgabe, die wiederum aus dem Aufruf von *UpdateWiimoteChanged* besteht. Im Normalfall erfolgt ein solcher Aufruf aus einer anderen Methode heraus synchron. Das bedeutet sie werden nacheinander ausgeführt und *wm_WiimoteChanged* wird erst wieder fortgesetzt wenn *UpdateState* und die angeschlossenen Methoden vollständig durchlaufen wurden. In *UpdateWiimoteChanged* jedoch sind Methoden aus der *Graphics*-Klasse enthalten, die bei einem solchen synchronen Aufruf zum Absturz der Systemfunktion führen. Daher kann *UpdateWiimoteChanged* nicht konventionell aufgerufen werden sondern muss über den Delegaten *UpdateWiimoteStateDelegate* aufgerufen werden, der wiederum mit dem *BeginInvoke* Befehl asynchron ausgeführt wird. *UpdateWiimoteChanged* läuft `somit` parallel zu den anderen Methoden ab und es kann die

Systemfunktion garantiert werden. Zusätzlich werden über das *args*-Argument die Wimote-Statusinformationen übergeben (siehe Anhang B, Seite 17).

private void UpdateWiimoteChanged(WiimoteChangedEventArgs args)

Diese werden in *UpdateWiimoteChanged* genutzt um eine Klasse *WiimoteState ws* zu instanzieren, mit der nun die Wiimote-Parameter weiterhin abgerufen werden können. Eine weitere wichtige Voreinstellung, ist die Festlegung der *PictureBox IRBox* als das *Bitmap b,* das bereits zu Anfang der Anwendung in der *MainForm*-Methode mit der Zeichenoberfläche *g* der *Graphics*-Klasse verknüpft wurde. Somit ist die Oberfläche der *PictureBox* nun fähig mit Hilfe von Zeichenmethoden aus der *Graphics*-Klasse grafische Elemente abzubilden. Sie dient als Ausgangspunkt für das weitere Vorgehen.

Um einen sich bewegenden Punkt darzustellen, der die IR-Signale wiederspiegelt, muss nach dem Einzeichnen der Punkte, das in der folgenden Methode *UpdateIR* geschieht, zuvor aber jedes Mal das alte Abbild der IR-Punkte wieder gelöscht werden. Dies geschieht mit der *g.Clear(Color.White)* Anweisung, die bei jedem Aufruf von *UpdateWiimoteChanged* die Zeichenoberfläche wieder weiß färbt und somit die alten Markierungen der IR-Positionen löscht. Für das Einzeichnen des Gitters und der Berandung, die die Oberfläche in die sechs Felder einteilen (siehe Abbildung 6.5 in Kapitel 7.3.1) sind die verschiedenen *g.DrawLine* Methoden verantwortlich, die, wie der Name bereits vermuten lässt, nur verschiedene Linien zeichnen.

Zudem wird, falls die *feedback*-Variable in der *wm_WiimoteChanged*-Methode nach Auslösen eines Samples den Wert *true* bekommen hat, das betätigte Feld durch Abfrage des momentan in der *note*-Variable gespeicherten Notenwerts identifiziert und mit dem *g.FillRectangle* Befehl über Angabe der Koordinaten der verschiedenen Felder grün gefärbt.

Ein weiteres Feature der GUI ist die Anzeige der Notenwerte in den Feldern durch einfache Schriftzüge. Diese sind durch den Designer der Windows Forms Application bereits auf der Bedienoberfläche implementiert. Um den Schriftzügen oder *labels*, wie sie im Designer auch genannt werden, allerdings immer die aktuellen *string*-Variablen zugrunde zu legen, müssen diese immer wieder in einer ständig durchlaufenen Methode wie *UpdateWiimoteChanged* überprüft werden. Dazu werden die Notenwert-Variablen abgerufen, anschließend zu einer *string*-Variablen konvertiert und schließlich als Text des gewünschten Labels festgelegt. Untenstehender Codeausschnitt zeigt ein Beispiel für diesen Prozess, das Feld A betrifft.

```
TextA = Convert.ToString(A);
TextNoteA.Text = TextA;
```

Damit die endgültige Zeichnung der IR-Punkte erfolgen kann, wird als letzer Schritt die dafür verantwortliche Methode *UpdateIR* aufgerufen. Ihr werden zusätzlich zwei notwenige Variablen übergeben. Zum einen mit *ws.IRState.IRSensors* die gesamten Statusinformationen des Infrarotsensors und zum anderen mit *Color* die definierte Farbe, abhängig davon welches der drei IR-Signale ausgewählt ist (siehe Anhang B, Seite 17).

`private void UpdateIR(IRSensor irSensor, Color color)`

Die Aufgabe dieser Methode besteht allein darin die IR-Punkte einzuzeichnen. Dafür sind ihr, wie soeben bereits erwähnt, die Daten des Infrarotsensors und die jeweilige Farbe als Variablen übergeben worden. Mit Hilfe einer *if*-Schleife wird über die Bedingungssyntax *irSensor.Found* zunächst überprüft ob ein jeweiliges Infrarotsignal überhaupt vorhanden ist. Bei positiver Resonanz diesbezüglich wird die *FillRectangle* – Methode der *Graphics*-Klasse *g* ausgeführt, die mit den ihr zur Verfügung gestellten Parametern der Koordinatenposition sowie den Werten zu Ausmaßen, ein ausgefülltes Rechteck zeichnet. Dabei werden Höhe und Breite jeweils als 4 Bildpunkte definiert, die Koordinaten entsprechen den x- und y-Positionen der IR-Signale, die mit dem Faktor 9/16 multipliziert werden damit die Infrarotkameraauflösung von 1024 zu 768 Pixel auf das Ausmaß des als Zeichenoberfläche verwendeten Bitmaps von 576 zu 432 Bildpunkten skaliert wird, um entsprechend abgebildet werden zu können (siehe Anhang B, Seite 18).

7.3.2.4.6. Timer-Methoden

`void timer1_Tick(object sender, EventArgs e)`

Die Funktion der vier Timer-Methoden beschränkt sich darauf, eine entsprechende Timer-Variable auf den boolschen *true*-Wert zu setzen (siehe Anhang B, Seite 18). Sie sind entweder Teil der Funktion, dem Doppeltauslösen eines Samples oder Switch-Befehls entgegenzuwirken, welche bereits im Kapitel Wiimote-Gestenerkennungsmethoden 7.3.2.4.4 ausführlich erklärt wurde oder beenden das Aufleuchten eines Feldes im Zusammenhang mit dem visuellen Feedback (siehe Kapitel 7.3.2.4.3). In beiden Fällen ist die Darlegung der Funktion bereits hinreichend beschrieben. Es sei noch darauf hingewiesen, dass alle Timer-Variablen und deren Intervalle in der Methode *Form1_Load* definiert wurden.

7.3.2.4.7. GUI-Methoden

Unter diesem Begriff werden alle Arten von Methoden verstanden, die ausgeführt werden, nachdem eine Eingabe über die Benutzeroberfläche, vor allem mit Maus und Tastatur, getätigt wird. Sie repräsentieren Funktionen, die durch diverse Schaltelemente des GUI ausgelöst werden können. Für die weiteren Erklärungen wird nochmals auf die Abbildungen 6.5 und 6.6 auf den Seiten 48 und 49 hingewiesen, die die gesamte Form der Software-Bedienoberfläche repräsentieren.

`private void mIDISetupToolStripMenuItem_Click(object sender, EventArgs e)`

Durch klicken auf den Punkt „MIDI-Setup“ im Optionsmenü in der Menüleiste am oberen Ende der GUI öffnet sich das *MidiOptions*-Fenster das mit der Klasse *MidiOptions* verknüpft ist und das wie bereits erwähnt von Wii-Drum-Synth übernommen wurde. Darum ist der Quelltext von *MidiOptions* zwar dem

Anhang B auf den Seiten 22-24 beigefügt, es wird darauf aber nicht näher eingegangen. Für das Verständnis der Funktion des Optionsmenüs ist die folgende Betrachtung ausreichend.

Zu Anfang wird eine neue Instanz *options* aus der *MidiOptions*-Klasse gebildet um auf deren Objekte zugreifen zu können. Danach werden die MIDI-Eigenschaften festgelegt. Im MIDI-Standard können für die Sendung der Daten verschiedene Kanäle (Channels) und Programme bzw. Instrumente festgelegt werden. Bei der Nutzung von MIDI-Befehlen mit Softwareinstrumenten in dieser Arbeit sind diese aber irrelevant und können völlig ignoriert werden. Dagegen ist die Einstellungsoption des MIDI-Ausgangsgeräts unter dem Punkt *OutputDevice* von essentieller Bedeutung. Unter dem Befehl *options.OutDeviceId = synth.OutputDeviceId* wird bei Aufruf des Menüs das momentane Outputgerät, das das *synth*–Objekt der *Synth*-Klasse des C# MIDI Toolkits verwendet, abgerufen. Mit Hilfe der Auswahlbox unter „Output Device" können nun jedoch auch andere MIDI-Verbindungselemente wie der interne LoopBe-Treiber, die hier automatisch durch nicht näher beschriebene Vorgänge in der *MidiOptions*-Klasse gezeigt werden, ausgewählt werden (siehe Abbildung 6.6). Bei Drücken des OK-Buttons nimmt das *synth*-Objekt das selektierte MIDI-Ausgangsgeräts an (siehe Anhang B, Seite 18-19):

```
if (options.ShowDialog() == DialogResult.OK)
{
    synth.OutputDeviceId = options.OutDeviceId;
}
```

private void trackBar1_Scroll(object sender, EventArgs e)

Ein weiteres Element der Software-Oberfläche ist die so genannte *trackbar*, am unteren rechten Ende des Hauptfensters platziert, über die sich die Mindest-Velocity einstellen lässt. Darunter ist ein Label platziert, das den momentanen Wert dazu anzeigt. Wird also der Regler in dem *trackbar*-Element verschoben, kommt es automatisch zu einem Aufruf der *trackBar1_Scroll*-Methode. Diese stellt dann die Variable *Velocity_min*, die in `wm_WiimoteChanged` verwendet wird, auf den momentanen Wert `trackBar1.Value` der *trackbar*. Um diesen in dem Schriftzug darunter wiederzugeben, muss die *integer*-Variable *Velocity_min* in den Typ *string* konvertiert werden, damit er dem Text des Labels *Label_Vel_min.Text* anschließend zugeordnet werden kann (siehe Anhang B, Seite 19)

private void numericUpDown1_ValueChanged(object sender, EventArgs e).

Als *numericUpDowns* werden die neun Schaltelemente bezeichnet, die zum Einen ihren momentanen Wert in einem kleinen Kästchen wiedergeben, dieser sich aber, entweder durch eintragen einer anderen Zahl mittels der Tastatur oder durch Klicken auf die Pfeiltasten zur Rechten der Kästchen, verändern lassen. In diesen Fällen werden die dazu passenden Methoden *numericUpDown_ValueChanged* aufgerufen.

Eines dieser Elemente ist für die Einstellung des Sensitivity Parameters *decimal* Sens zuständig, der wiederum in die Berechnung der Beschleunigungswerte in y-Achsenrichtung mit einfließt, um die Auslösung der Samples auch zu gewährleisten, wenn der Nutzer das System in verschiedene Entfernungen zur IR-Kamera verwendet. Dazu wird der *Sens*-Variable einfach der Wert *numericUpDown1.Value* des *numericUpDowns* übergeben (siehe Anhang B, Seite 19).

private void numericUpDown3_ValueChanged(object sender, EventArgs e)

Die weiteren acht *numericUpDown*-Schaltelemente sind jeweils nach und vor jedem Samplepad um die Fläche herum angeordnet auf die die IR-Punkte projiziert werden. Sie definieren wie in Kapitel 7.3.1 bereits angedeutet die oberen und unteren Grenzen der Notenwerte für die sechs Pads. Die dadurch festgelegten Bereiche für die jeweiligen Notenwerte sollen zwar immer aneinander liegen, sich aber nie überschneiden. Als Beispiel für alle acht *numericUpDowns* wird die Syntax zur Methode *numericUpDown3_ValueChanged* erklärt, die den Grenzwert zwischen Pad A und B definiert.

Um die soeben genannten Funktionen zu gewährleisten, sind vier zu überprüfende Bedingungen erforderlich, die alle mit einem *if*-Befehl abgefragt werden. Mit den ersten beiden wird garantiert, dass der Wert *numericUpDown3.Value* des *NumericUpDowns3* sich nicht überschneidet mit den Werten *numericUpDown2.Value* von *NumericUpDown2* und *numericUpDown4.Value von NumericUpDown4*. Bedingung drei und vier stellen sicher, dass der Notenwert des Pads A *numericUpDown3.Value* nicht überschreitet und der Notenwert des Pads B diesen nicht unterschreitet (siehe Anhang B, Seite 19-20).

Alle Methoden zu den anderen *NumericUpDowns* sind gleich implementiert. Nur für die Schaltelemente vor Pad A und nach Pad F entfallen jeweils zwei der Bedingungen, da diese beiden nur die Grenze für ein statt zwei Pads darstellen (siehe Anhang B, Seite 19-22).

private void Form1_FormClosing(Object sender, FormClosingEventArgs e)

Form1_FormClosing wird bei der Betätigung des roten X-Schaltelements ausgeführt, das in jedem Windows Fenster neben den Schaltflächen für Maximieren und Minimieren in der oberen rechten Ecke sitzt. Diese Methode schließt und beendet die gesamte Anwendung und trennt mit dem Befehl *wm.Disconnect* die Verbindung von Wiimote zu Wii Infrared MIDI.

Mit diesem Kapitel endet die Beschreibung der einzelnen Methoden.

7.3.3. Benutzerhandbuch

Als Abschluss des Wii Infrared MIDI Kapitels soll an dieser Stelle eine knappe, aber dennoch deutliche Installations- sowie Bedienungsanleitung beschrieben werden, mit deren Hilfe der Nutzer auch ohne Kenntnis der soeben ausgeführten Syntax und aller anderer in dieser Arbeit bereits formulierten Kapitel das Gesamtinterface anwenden kann. Das Programm wurde unter Windows 7 erstellt und getestet, kann aber auch unter anderen Windows Versionen verwendet werden.

7.3.3.1. Installationsanleitung

Durch die eingesetzte .NET-Technologie gestaltet sich die Installation relativ einfach. Voraussetzung ist hierbei natürlich, dass das .NET-Framework 4.0 auf dem gewünschten PC bereits installiert ist. Ob dies der Fall ist, kann in der Systemsteuerung unter Programme/Programme und Funktionen nachgesehen werden. Ist dies nicht der Fall, so kann das Framework in der neuesten Version unter http://msdn.microsoft.com/ unter der Rubrik „Downloads“ kostenfrei heruntergeladen werden.

Nach diesem Schritt muss nur noch der Ordner „Wii Infrared MIDI“, der der Bachelor-Arbeit beiliegt in ein beliebiges Verzeichnis kopiert werden und die Setup Datei ausgeführt werden. Damit ist die Installation abgeschlossen.

7.3.3.2. Bedienungsanleitung

Vor dem Programmstart muss die Wiimote mit dem PC per Bluetooth verbunden werden. Dafür wird ein Bluetooth-Empfänger benötigt, der nicht in allen Computern eingebaut ist. Im Systemsteuerungsmenü unter Hardware und Sound/Geräte und Drucker/Bluetooth-Geräte kann kontrolliert werden, ob ein solches Gerät vorhanden ist. Oftmals ist aber ein zusätzlicher Bluetooth-Adapter notwendig, der für maximal etwa 5 Euro in jedem gut sortierten Elektronikfachmarkt zu kaufen sein sollte. Es empfiehlt sich weiterhin eine spezielle Software für den Verbindungsaufbau zwischen Wiimote und PC zu verwenden. Ein Beispiel hierfür wäre das in Kapitel 4.2.2 gezeigte Programm „Blue Soleil“. Anschließend muss nach Bluetooth-Geräten gesucht werden. Diese Funktion sollte in jeder Bluetooth-Software vorhanden sein. Zur Erkennung der Wiimote müssen die Tasten 1 und 2 gleichzeitig gedrückt werden, damit die Wii-Remote ein entsprechendes Signal sendet. Dabei blinken die vier LEDs an der Bedienungseinheit. Sie wird anschließend als Human Interface Device mit dem Namen „Nintendo RVL-CNT-01“ erfasst. Nun sollte die verwendete Bluetooth-Software eine Option anbieten, die ein Verbinden der Wiimote mit dem PC ermöglicht.

Der Programmstart erfolgt über einen Doppelklick der Datei Wiimote Infrared MIDI in dem Verzeichnis, das zur Installation gewählt wurde. Falls keine Verbindung der Wiimote zur Software besteht, wird dies beim Start direkt durch die Fehlermeldung „No Wiimotes found in HID device list“ des .NET Frameworks angezeigt. In diesem Fall ist es ein probates Mittel, die Wiimote vom PC zu trennen, erneut anzuschließen und dann noch einmal Wii Infrared MIDI zu starten. Die Wiimote muss zur Funktion des Systems unbedingt

auf der Knopfleiste platziert werden (siehe Abbildung 6.7 auf Seite 52). Die Gründe dafür wurden in Kapitel 7.3.2.2 eingehend erläutert.

Nach dem Start der Anwendung sollte die Software-Oberfläche der untenstehenden Abbildung 7.8 gleichen. Zunächst sollte die gewünschte MIDI-Verbindung im Options-Menü unter „Output Device" ausgewählt werden und mit dem OK-Button bestätigt werden. Die anderen Optionsmöglichkeiten sind hier irrelevant. Um die MIDI-Befehle in dem Software-Instrument zu empfangen muss in dem virtuellen Klangerzeuger natürlich auch das MIDI-Verbindungselement als MIDI-Input ausgewählt werden. An der grünen Leiste am unteren Ende des Graphical User Interface kann die verbleibende Batterieleistung abgelesen werden.

Die mittlere Fläche, die in sechs Rechtecke eingeteilt ist, bildet das visuelle Hauptrückgabeelement. Hiermit können bis zu drei Infrarotsignale wiedergegeben werden und somit die Infrarotbewegungen koordiniert werden. Prinzipiell können dabei alle Infrarotlichtquellen verwendet werden, diese sollten aber in ihren Bewegungen gut kontrolliert werden können, wie etwa Handschuhe mit eingearbeiteten LEDs.

Durch eine schnelle Bewegung nach unten erfolgt nun die Betätigung eines MIDI-Befehls, dessen Velocity-Wert von der Intensität der Dynamik abhängt. Es kann aber ein Mindestwert dieser Velocity an dem in der Abbildung 7.8 gelb eingekreiste Regler erfolgen. Sobald die Bewegung erfolgreich ausgeführt wird, leuchtet das betätigte Feld kurz grün auf.

Zusätzlich kann die Sensibilität des Systems hinsichtlich der Infrarotbewegungen durch den blau markierten Sensitivity-Faktor angepasst werden, damit die Nutzung des Systems in verschiedenen Entfernungen der IR-Signale von der Wiimote gewährleistet ist. Für einen Sensitivity-Faktor von Null ist etwa ein Meter die optimale Entfernung. Von kleineren Distanzen zwischen der IR-Bewegung und der Wiimote wird abgeraten.

Die Länge der gesendeten MIDI-Note ist allerdings nicht steuerbar. Dieser ist vorgegeben und immer von kurzer Dauer. Daher bietet es sich auch an, das Interface für virtuelle Musiksoftware an, die mit Samples arbeitet, da diese, im Gegensatz zu beispielsweise virtuellen Klavier- und Streicher-Instrumente, die auf das Halten von Noten und Akkorden angewiesen sind, auch bei kurzen Befehlen ausgelöst und komplett abgespielt werden.

Je nachdem in welchem der Felder die Bewegung ausgeführt wird, sendet das Interface einen anderen MIDI-Notenwert. In den einzelnen Rechtecken wird dieser Wert, in der Abb. 7.8 grün gekennzeichnet, angezeigt. Mit dem so genannten Switch-Befehl können diese Notenwerte gewechselt werden. Dazu muss ein Infrarotpunkt mit einer Auswahlbewegung auf das zu wechselnde Feld zeigen, während die anderen beiden, egal in welchem Feld, jedoch gleichzeitig und untereinander eine schnelle Bewegung nach rechts für die Switch-Forward-Befehl oder nach links für die Switch-Back-Befehl ausführen müssen. Dadurch wird der Notenwert des ausgewählten Feldes um einen Schritt erhöht (Switch-Forward) oder erniedrigt (Switch-Back).

Die Notenwerte können aber immer nur zwischen den rot gekennzeichneten, einstellbaren Notenwertgrenzen liegen, damit es zu keine Sample-Überschneidungen kommt.

Zur Verwendung der Demoversion des virtuellen Samplers Battery 3 und der Sequenzing Software Mulab 3 wird auf die Hilfe-Dokumentationen in den Programmen verwiesen.

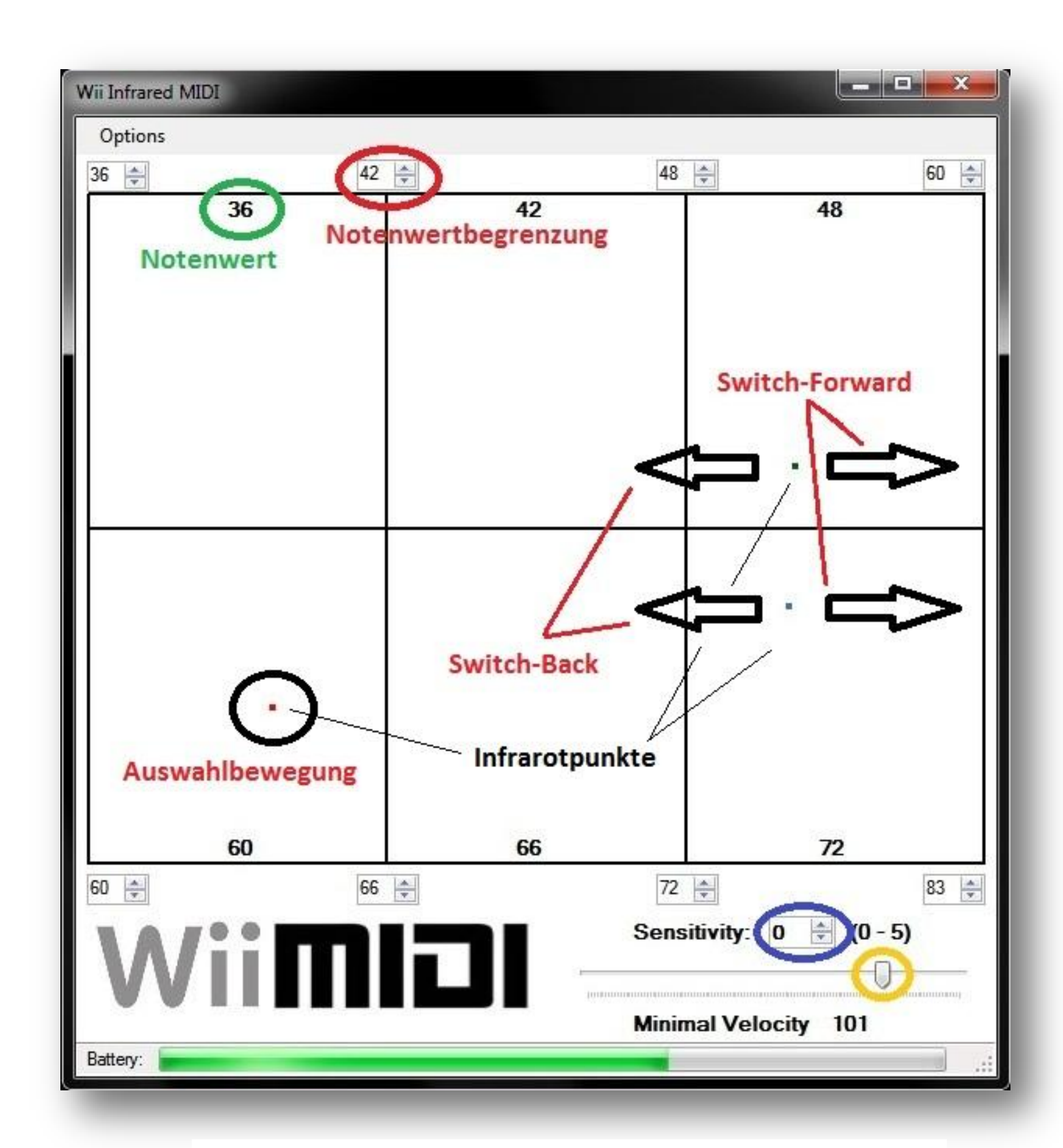


Abb. 7.8: Wii Infrared GUI mit Veranschaulichungen

8. Ausblick

Das Wii-Infrared-MIDI-Interface, das in dieser Arbeit vorgestellt wurde, ist sicherlich von Produkten mit industriellem Standard weit entfernt, es zeigt jedoch, dass es möglich ist, ein Interface zu entwickeln, dass in der Lage ist eine menschliche Bewegung direkt in die Erzeugung eines Klangs umzusetzen und somit durchaus als intuitive Eingabeform verstanden werden kann, obwohl diese nur über den Umweg von Infrarotsignalen erfolgt. Die Erkennung von bestimmten Gesten um Musik zu erzeugen bzw. zu steuern ist dabei aber ein sinnvolles Mittel, da hiermit eine unmittelbar Zugriffsmöglichkeit auf die Variation verschiedener Parameter besteht.

Eine Weiterentwicklung wäre beispielsweise die Wahrnehmung der gesamten menschlichen Dynamik als Eingabeform. Die Kinect-Technologie, die für die Spielekonsole „Xbox 360" der Firma Microsoft entwickelt wurde, zeigt hier die Richtung an. Sie kann durch zwei Kameras ein räumliches Bild erzeugen, das den Menschen in seinen Umrissen erkennt, seine Bewegungen auch in der Tiefe erfasst und alle diese Information theoretisch weiterverarbeiten kann. In Kapitel 2.3.3.1 wurde bereits auf ein Video zur Steuerung eines PC-Betriebssystems mit der Technologie hingewiesen. Abbildung 8.1 zeigt eine Visualisierung eines Bildes aus der Kinect-Sicht mit verschiedenen Farbstufen für unterschiedliche Entfernungen der jeweiligen Objekte.



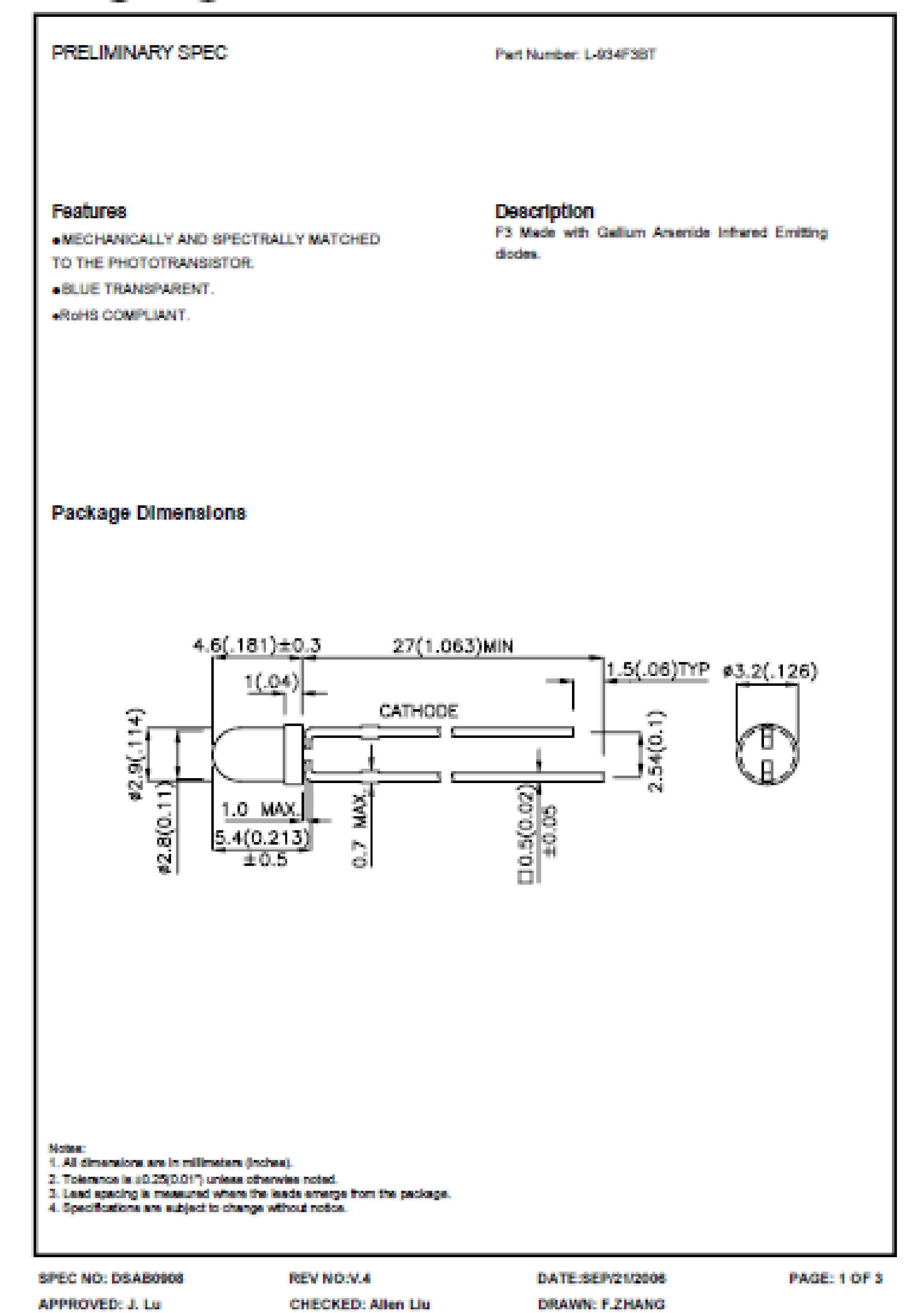
Abb. 8.1: Visualisierung der Kinect-Raumdaten

Angelehnt an das Funktionsprinzip des in Kapitel ~~XXX~~ beschriebenen Reactable, könnten ebenfalls Objekte im Raum platziert werden, die etwa als Klangerzeuger oder Modulator dienen. Je nach Position ändern sich dabei ihre Funktion oder die beeinflussenden Parameter ab. Verschiedene der Komponenten sind dabei kombinierbar und verändern ihre musikalische Gestalt im Zusammenspiel. Dabei könnten die Objekte auch virtueller Natur sein und nur über Bildschirme oder grafische Projektionen visualisierbar werden.

Mit dieser Art Eingabegerät für die Computer-Musikproduktion wäre der Zugang noch direkter, da ohne den Umweg über Infrarotleuchtdioden die eigentlich menschliche Bewegung direkt als Input für die Software dienen kann und es wäre ähnlich der Spielweise eines klassischen Instruments eine unmittelbare Verbindung zwischen menschlicher Mechanik und Technik geschaffen.

9. Anhang

9.1. Anhang A: Datenblatt Kingbright-Infrarot-Leuchtdiode



Kingbright

T-1 (3mm) INFRA-RED EMITTING DIODE

PRELIMINARY SPEC

Part Number: L-934F3BT

Features

- MECHANICALLY AND SPECTRALLY MATCHED TO THE PHOTOTRANSISTOR.
- BLUE TRANSPARENT.
- RoHS COMPLIANT.

Description

F3 Made with Gallium Arsenide Infrared Emitting diodes.

Package Dimensions

Notes:
1. All dimensions are in millimeters (inches).
2. Tolerance is ±0.25(0.01") unless otherwise noted.
3. Lead spacing is measured where the leads emerge from the package.
4. Specifications are subject to change without notice.

SPEC NO: DSAB0908 | REV NO:V.4 | DATE:SEP/21/2006 | PAGE: 1 OF 3
APPROVED: J. Lu | CHECKED: Allen Liu | DRAWN: F.ZHANG

Kingbright

Selection Guide

Part No.	Dice	Lens Type	Po(mW/sr) [2] @ 20mA*50mA		Viewing Angle [1]
			Min.	Typ.	2θ1/2
L-934F3BT	GaAs	BLUE TRANSPARENT	7	28	50°
			*18	*70	50°

Notes:
1.θ1/2 is the angle from optical centerline where the luminous intensity is 1/2 the optical centerline value.
2. * Luminous intensity with asterisk is measured at 50mA; Radiant intensity/ luminous flux: +/-15%.

Electrical / Optical Characteristics at T_A=25°C

Item	P/N	Symbol	Typ.	Max.	Units	Test Conditions
Forward Voltage [1]	F3	V_F	1.2	1.6	V	I_F=20mA
Reverse Current	F3	I_R	-	10	uA	V_R=5V
Capacitance	F3	C	90	-	pF	V_F=0V;f=1MHz
Peak Spectral Wavelength	F3	λP	940	-	nm	I_F=20mA
Spectral Bandwidth	F3	Δλ1/2	50	-	nm	I_F=20mA

Note:
1. Forward Voltage: +/-0.1V.

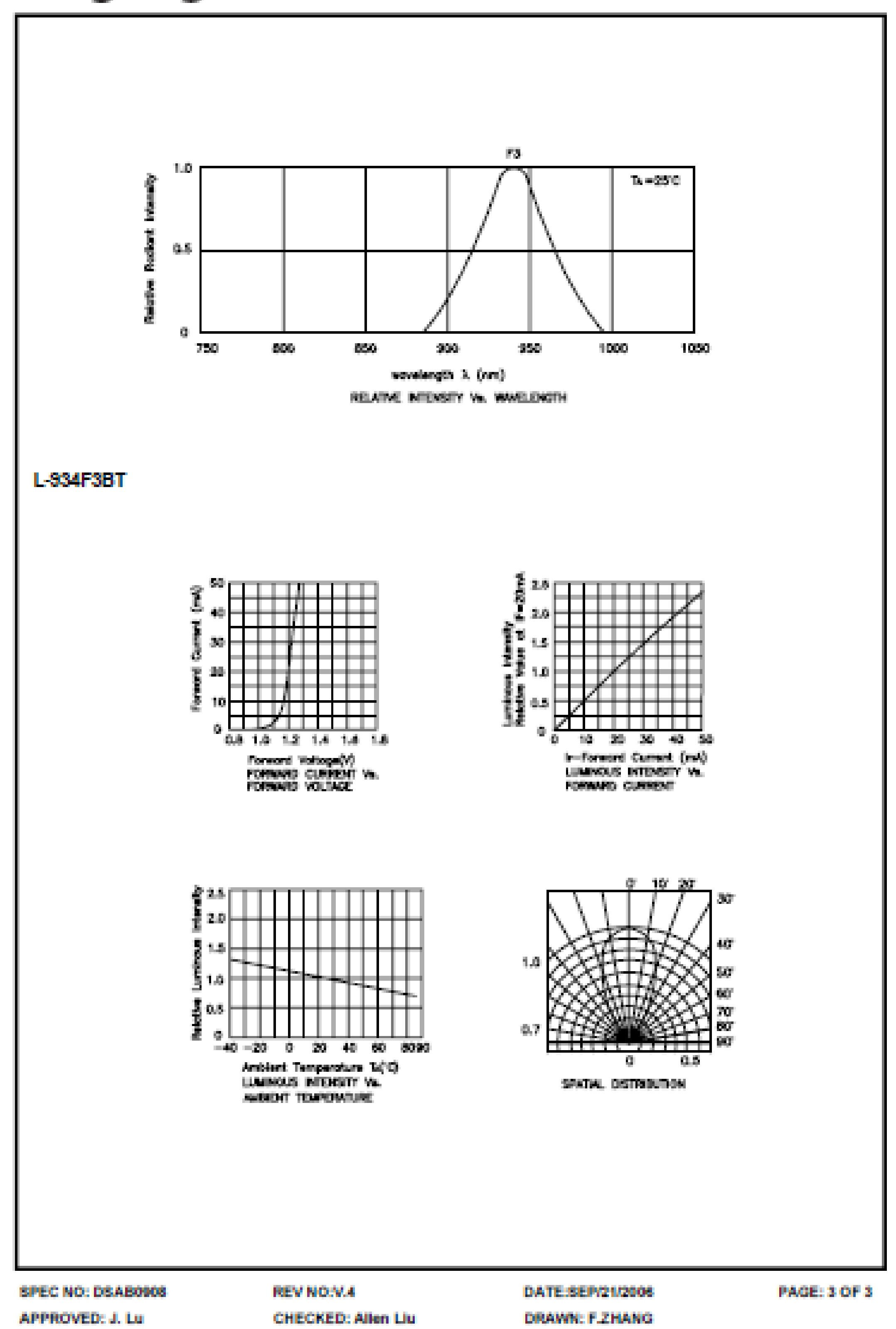
Absolute Maximum Ratings at T_A=25°C

Parameter	Symbol	F3	Units
Power Dissipation	P_T	100	mW
DC Forward Current	I_F	50	mA
Peak Forward Current[1]	I_{FS}	1.2	A
Reverse Voltage	V_R	5	V
Operating Temperature	T_A	-40 To +85	°C
Storage Temperature	T_{STG}	-40 To +85	°C
Lead Solder Temperature [2]	260°C For 3 Seconds		
Lead Solder Temperature [3]	260°C For 5 Seconds		

Notes:
1. 1/100 Duty Cycle, 10ms Pulse Width.
2. 2mm below package base.
3. 5mm below package base.

SPEC NO: DSAB0908 REV NO:V.4 DATE:SEP/21/2006 PAGE: 2 OF 3
APPROVED: J. Lu CHECKED: Allen Liu DRAWN: F.ZHANG

Kingbright



L-934F3BT

SPEC NO: DSAB0908 REV NO:V.4 DATE:SEP/21/2006 PAGE: 3 OF 3

APPROVED: J. Lu CHECKED: Allen Liu DRAWN: F.ZHANG

9.2. Anhang B: Quelltext

Alle Kommentare stehen unmittelbar über dem jeweiligen Code.

```
////////////////////////////////////////////////////////////////////////////////////////
//
//  Bachelorarbeit: Making music using intuitive interfaces
//
//  Fabian Seipel
//
//  Unter Verwendung von:
//  - WiimoteLib (Brian Peek), www.brainpeek.com
//  - Wii Drum Synth, wiidrumsynth.codeplex.com
//  - C# MIDI Toolkit (Leslie Sanford),
//    www.codeproject.com/KB/audio-video/MIDIToolkit.aspx
//
////////////////////////////////////////////////////////////////////////////////////////
```

9.2.1. Class MainForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using WiimoteLib;
using System.Diagnostics;
using System.Media;
using Sanford.Multimedia.Midi;
using System.IO;
using System.Collections;
using System.Globalization;
using System.Threading;
using MidiSynthLib;


namespace WindowsFormsApplication1
{
    public partial class MainForm : Form
    {

        // Neue Instanz des Wiimote-Objekts aus der WiimoteLib erstellen
        Wiimote wm = new Wiimote();

        // Neue Instanz der Klasse Synth aus der MidiSynthLib erstellen
        private Synth synth;


        // Deklaration des Delegats als Grundlage für das UpdateWiimoteChanged Event
        private delegate void
        UpdateWiimoteStateDelegate(WiimoteChangedEventArgs args);
```

```
////////// Definition aller globalen Variablen

        // Übergabe-Variablen der y-Positionen
        float Y1_vorher;
        float Y2_vorher;
        float Y3_vorher;

        // Variablen der Geschwindigkeit in y-Richtung
        float V1_y;
        float V2_y;
        float V3_y;

        // Übergabe-Variablen der Geschwindigkeit in y-Richtung
        float V1_y_vorher;
        float V2_y_vorher;
        float V3_y_vorher;

        // Variablen der Beschleunigung in y-Richtung
        float A1_y;
        float A2_y;
        float A3_y;

        // Übergabe-Variablen der x-Positionen
        float X1_vorher;
        float X2_vorher;
        float X3_vorher;

        // Variablen der Geschwindigkeit in x-Richtung
        float V1_x;
        float V2_x;
        float V3_x;

        // Variablen der Positionsdifferenzen in x-Richtung
        float X_Gap12;
        float X_Gap23;
        float X_Gap13;

        // Zeitvariable zur Bildung der Differenzenquotienten
        float t = 10;

        // Mindestwert der y-Beschleunigung zum Auslösen der Samples
        float A_y_Thres = 4;
        // Maximalwert der y-Beschleunigung zum Auslösen der Samples
        float A_y_Max = 20;
        // Mindestwert der x-Geschwindigkeit zum Auslösen des Switch-Forward-Befehls
        float V_x_Thres_plus = 2;
        // Mindestwert der x-Geschwindigkeit zum Auslösen des Switch-Back-Befehls
        float V_x_Thres_minus = -2;
        // Maximalwert der x-Geschwindigkeit zum Auslösen des Switch-Forward-Befehls
        float V_x_Max = 20;
        // Maximalwert der x-Geschwindigkeit zum Auslösen des Switch-Back-Befehls
        float V_x_Min = -20;
        // Maximalwert der Positionsdifferenzen zum Azslösen der Switch-Befehle
        int X_Gap_Max = 150;

        // allgemeine Notenwert-Variable
        int note;
        // Variablen der Notenwerte der einzelnen Felder
        int A = 36;
        int B = 42;
        int C = 48;
        int D = 60;
```

```
        int E = 66;
        int F = 72;

        // Sensitivity-Variable
        decimal Sens = 0;

        // Variable zur Mindest-Veocity
        int Velocity_min = 120;

        // Variablen zur Anzeige der Notenwerte auf der GUI
        string TextA;
        string TextB;
        string TextC;
        string TextD;
        string TextE;
        string TextF;

        // Variable zur Anzeige der Mindest-Velocity auf der GUI
        string Text_Vel_min;

        // Erstellen der Timer
        static System.Windows.Forms.Timer timer1 = new System.Windows.Forms.Timer();
        static System.Windows.Forms.Timer timer2 = new System.Windows.Forms.Timer();
        static System.Windows.Forms.Timer timer3 = new System.Windows.Forms.Timer();
        static System.Windows.Forms.Timer timer4 = new System.Windows.Forms.Timer();
        static System.Windows.Forms.Timer timer5 = new System.Windows.Forms.Timer();
        int all_timer_interval = 100;

        // Timer-Variablen
        bool peak_filter_Y1 = true;
        bool peak_filter_Y2 = true;
        bool peak_filter_Y3 = true;
        bool peak_filter_X = true;
        bool feedback = false;

        // Definiert eine Zeichenoberfläche, die zur Anzeige der IR-Signale dient
        private Graphics g;

        // Definiert die Bitmap-Bilddatei, auf der gezeichnet wird
        private Bitmap b = new Bitmap(576, 432, PixelFormat.Format24bppRgb);


////////// Methoden

        // Methode, die bei Programmstart aufgerufen wird
        public MainForm()
        {
            // Erforderliche Methode für die Designerunterstützung
            InitializeComponent();

            // Definiert das Synth-Objekt aus der Synth-Klasse,
            // das weiter unten zum Senden der MIDI-Befehle dient
            synth = new MidiSynthLib.Synth(9, 118);

            // Definiert die Bilddatei b als Zeichenoberfläche g
            g = Graphics.FromImage(b);


        ////// Mindest-Velocitywert auf die GUI projizieren

            // Integer-Variable A des Notenwerts im Feld A
            // in String-Variable TextA konvertieren
            Text_Vel_min = Convert.ToString(Velocity_min);
```

```
        // String Variable Text_Vel_min als Label Schriftzug_Vel_min
        // der GUI definieren
        label_Vel_min.Text = Text_Vel_min;


        // Stellt sicher, dass zu Anfang der Anwendung die Werte von
        // numericUpDown5 und numericUpDown6 gleich sind
        numericUpDown6.Value = numericUpDown5.Value;

    }

    // Methode, die bei Aufrufen des Wii Infrared MIDI Fesnsters aufgerufen wird
    //
    private void Form1_Load(object sender, EventArgs e)
    {
        // Erstellt den Eventhandler zur Methode wm_WiimoteChanged
        wm.WiimoteChanged
        += new EventHandler<WiimoteChangedEventArgs>(wm_WiimoteChanged);

        // Erstellt den Eventhandler zur Methode wm_WiimoteChanged_Switch
        wm.WiimoteChanged
        += new EventHandler<WiimoteChangedEventArgs>(wm_WiimoteChanged_Switch);

        // Verbindet die Wiimote
        wm.Connect();

        // Definiert welche LEDs an der Wiimote leuchten
        wm.SetLEDs(false, false, true, true);

        // Definiert den Datenreporttyp und die Sensivität der Infrarotkamera
        wm.SetReportType(InputReport.IRAccel, IRSensitivity.Maximum, true);

        // Legt den Datenreporttyp der IR Kamera fest
        wm.WiimoteState.IRState.Mode = IRMode.Extended;

    ////// Batteriestatus auf die GUI projizieren

        // Ruft den Batteriestatus ab und überführt diesen in die float Variable B
        float B = wm.WiimoteState.Battery;
        // Stellt aus der float Variable B die integer Variable Bat her
        int Bat = (int)B;
        // Integer Variable Bat als Wert der ProgressBar1 auf der GUI definieren
        toolStripProgressBar1.Value = Bat;

    ////// Erstellen der Timer

        // Timer Event erstellen
        timer1.Tick += new EventHandler(timer1_Tick);
        // Festlegen des Zeitintervalls
        timer1.Interval = all_timer_interval;
        // Aktivieren des Timers
        timer1.Enabled = true;

        // Erstellung der anderen Timer analog zu Timer 1
        // (bis auf Intervalle des Timers 4 und 5)

        timer2.Tick += new EventHandler(timer2_Tick);
        timer2.Interval = all_timer_interval;
        timer2.Enabled = true;

        timer3.Tick += new EventHandler(timer3_Tick);
        timer3.Interval = all_timer_interval;
```

```
            timer3.Enabled = true;

            timer4.Tick += new EventHandler(timer4_Tick);
            timer4.Interval = 800;
            timer4.Enabled = true;

            timer5.Tick += new EventHandler(timer5_Tick);
            timer5.Interval = 50;
            timer5.Enabled = true;

        }



      //// Methoden zur Berechnung der Geschwindigkeiten der IR-Signale in y-Richtung

        // Berechnung IR1-Geschwindigkeit in y-Richtung
        private void berechne_V1_y()
        {
            // Y-Positionsdaten von der Wiimote abrufen
            float Y1 = wm.WiimoteState.IRState.IRSensors[0].RawPosition.Y;
            // Berechnung der Geschwindigkeit durch angenäherten Differenzenquotient
            V1_y = (Y1 - Y1_vorher) / t;
            // Überschreibung der globalen Variable Y1_vorher
            Y1_vorher = Y1;
        }

        // Berechnung IR2-Geschwindigkeit in y-Richtung (analog zu IR 1)
        private void berechne_V2_y()
        {
            float Y2 = wm.WiimoteState.IRState.IRSensors[1].RawPosition.Y;
            V2_y = (Y2 - Y2_vorher) / t;
            Y2_vorher = Y2;
        }

        // Berechnung IR3-Geschwindigkeit in y-Richtung (analog zu IR 1)
        private void berechne_V3_y()
        {
            float Y3 = wm.WiimoteState.IRState.IRSensors[2].RawPosition.Y;
            V3_y = (Y3 - Y3_vorher) / t;
            Y3_vorher = Y3;
        }


      //// Methoden zur Berechnung der Beschleunigungen der IR-Signale in y-Richtung

        // Berechnung IR1-Beschleunigung in y-Richtung
        private void berechne_A1_y()
        {
            // Aufruf der Methode zur Geschwindigkeitsbestimmung in y-Richtung
            berechne_V1_y();

            // Berechnung der Beschleunigung durch angenäherten Differenzenquotient
            A1_y = (((V1_y - V1_y_vorher) / t) +
                   (((V1_y - V1_y_vorher) / t) * (int)Sens * 1 / 5)) * 50;

            // Filter von Bewegungen in positive y-Richtung
            if (V1_y < 0)
                A1_y = 0;

            // Filtern der negativen Beschleunigungswerte
            if (A1_y < 0)
                A1_y = 0;
```

```
        // Überschreiben der globalen Variable V1_y_vorher
        V1_y_vorher = V1_y;
    }

    // Berechnung IR2-Beschleunigung in y-Richtung (analog zu IR 1)
    private void berechne_A2_y()
    {
        berechne_V2_y();

        A2_y = (((V2_y - V2_y_vorher) / t) +
               (((V2_y - V2_y_vorher) / t) * (int)Sens * 1 / 5)) * 50;

        if (V2_y < 0)
            A2_y = 0;

        if (A2_y < 0)
            A2_y = 0;

        V2_y_vorher = V2_y;
    }

    // Berechnung IR3-Beschleunigung in y-Richtung (analog zu IR 1)
    private void berechne_A3_y()
    {
        berechne_V3_y();

        A3_y = (((V3_y - V3_y_vorher) / t) +
               (((V3_y - V3_y_vorher) / t) * (int)Sens * 1 / 5)) * 50;

        if (V3_y < 0)
            A3_y = 0;

        if (A3_y < 0)
            A3_y = 0;

        V3_y_vorher = V3_y;
    }


  //// Methoden zur Berechnung der Geschwindigkeiten der IR-Signale in x-Richtung

    // Berechnung IR1-Geschwindigkeit in x-Richtung
    private void berechne_V1_x()
    {
        // X-Positionsdaten von der Wiimote abrufen
        float X1 = wm.WiimoteState.IRState.IRSensors[0].RawPosition.X;
        // Berechnung der Geschwindigkeit durch angenäherten Differenzenquotient
        V1_x = (X1 - X1_vorher) / t;
        // Überschreibung der globalen Variable X1_vorher
        X1_vorher = X1;
    }

    // Berechnung IR2-Geschwindigkeit in x-Richtung (analog zu IR 1)
    private void berechne_V2_x()
    {
        float X2 = wm.WiimoteState.IRState.IRSensors[1].RawPosition.X;
        V2_x = (X2 - X2_vorher) / t;
        X2_vorher = X2;
        //Debug.WriteLine("V2_x = " + V2_x);
    }
```

```
        // Berechnung IR3-Geschwindigkeit in x-Richtung (analog zu IR 1)
        private void berechne_V3_x()
        {
            float X3 = wm.WiimoteState.IRState.IRSensors[2].RawPosition.X;
            V3_x = (X3 - X3_vorher) / t;
            X3_vorher = X3;
        }


      //// Methoden zur Berechnung der Positionsdifferenzen in x-Richtung

        // Berechnung der Differenz zwischen Infrarotsignal 1 & 2
        private void berechne_X_Gap12()
        {
            // X-Positionsdaten von INfrarotsignal 1 & 2 der Wiimote abrufen
            float X1 = wm.WiimoteState.IRState.IRSensors[0].RawPosition.X;
            float X2 = wm.WiimoteState.IRState.IRSensors[1].RawPosition.X;

            // Differenz der Positionsdaten berechnen
            X_Gap12 = X1 - X2;

            // Betrag der Differenz bilden
            if (X_Gap12 < 0)
                X_Gap12 = X_Gap12 * (-1);
        }

        // Berechnung der Differenz zwischen Infrarotsignal 2 & 3 (analog zu 1 & 2)
        private void berechne_X_Gap23()
        {
            float X2 = wm.WiimoteState.IRState.IRSensors[1].RawPosition.X;
            float X3 = wm.WiimoteState.IRState.IRSensors[2].RawPosition.X;

            X_Gap23 = X2 - X3;
            if (X_Gap23 < 0)
                X_Gap23 = X_Gap23 * (-1);
        }

        // Berechnung der Differenz zwischen Infrarotsignal 1 & 3 (analog zu 1 & 2)
        private void berechne_X_Gap13()
        {
            float X1 = wm.WiimoteState.IRState.IRSensors[0].RawPosition.X;
            float X3 = wm.WiimoteState.IRState.IRSensors[2].RawPosition.X;

            X_Gap13 = X1 - X3;
            if (X_Gap13 < 0)
                X_Gap13 = X_Gap13 * (-1);
        }


      //// Methoden zum Auslösen der Switch-Forward-Befehle

        // Switch-Forward-Befehl für Feld A
        private void A_Switch_fwd()
        {
            // MIDI-Halbton zur Note des Feldes addieren
            A = A + 1;

            // Obergrenze für Notenwert des Feldes festlegen
            if (A > (int)numericUpDown3.Value - 1)
                A = (int)numericUpDown3.Value - 1;

        }
        // Switch-Forward-Befehl für Feld B (analog zu Feld A)
```

```
        private void B_Switch_fwd()
        {
            B = B + 1;
            if (B > (int)numericUpDown4.Value - 1)
                B = (int)numericUpDown4.Value - 1;
        }
        // Switch-Forward-Befehl für Feld C (analog zu Feld A)
        private void C_Switch_fwd()
        {
            C = C + 1;
            if (C > (int)numericUpDown5.Value - 1)
                C = (int)numericUpDown5.Value - 1;
        }
        // Switch-Forward-Befehl für Feld D (analog zu Feld A)
        private void D_Switch_fwd()
        {
            D = D + 1;
            if (D > (int)numericUpDown7.Value - 1)
                D = (int)numericUpDown7.Value - 1;
        }
        // Switch-Forward-Befehl für Feld E (analog zu Feld A)
        private void E_Switch_fwd()
        {
            E = E + 1;
            if (E > (int)numericUpDown8.Value - 1)
                E = (int)numericUpDown8.Value - 1;
        }
        // Switch-Forward-Befehl für Feld F (analog zu Feld A)
        private void F_Switch_fwd()
        {
            F = F + 1;
            if (F > (int)numericUpDown9.Value)
                F = (int)numericUpDown9.Value;
        }

      //// Methoden zum Auslösen der Switch-Back-Befehle

        // Switch-Back-Befehl für Feld A
        private void A_Switch_back()
        {
            // MIDI-Halbton von der Note des Feldes subtrahieren
            A = A - 1;

            // Untergrenze für Notenwert des Feldes festlegen
            if (A < (int)numericUpDown2.Value)
                A = (int)numericUpDown2.Value;

        }
        // Switch-Back-Befehl für Feld B (analog zu Feld A)
        private void B_Switch_back()
        {
            B = B - 1;
            if (B < (int)numericUpDown3.Value)
                B = (int)numericUpDown3.Value;
        }
        // Switch-Back-Befehl für Feld C (analog zu Feld A)
        private void C_Switch_back()
        {
            C = C - 1;
            if (C < (int)numericUpDown4.Value)
                C = (int)numericUpDown4.Value;
        }
```

```
// Switch-Back-Befehl für Feld D (analog zu Feld A)
private void D_Switch_back()
{
    D = D - 1;
    if (D < (int)numericUpDown5.Value)
        D = (int)numericUpDown5.Value;
}
// Switch-Back-Befehl für Feld E (analog zu Feld A)
private void E_Switch_back()
{
    E = E - 1;
    if (E < (int)numericUpDown7.Value)
        E = (int)numericUpDown7.Value;
}
// Switch-Back-Befehl für Feld F (analog zu Feld A)
private void F_Switch_back()
{
    F = F - 1;
    if (F < (int)numericUpDown8.Value)
        F = (int)numericUpDown8.Value;
}



// Methode zum Auslösen der Samples, die ständig von neuem durchlaufen wird,
// sobald die Wiimote Daten sendet
void wm_WiimoteChanged(object sender, WiimoteChangedEventArgs e)
{

    // Positionsdaten der IR-Signale der Wiimote abrufen
    float Y1 = e.WiimoteState.IRState.IRSensors[0].RawPosition.Y;
    float X1 = e.WiimoteState.IRState.IRSensors[0].RawPosition.X;
    float Y2 = e.WiimoteState.IRState.IRSensors[1].RawPosition.Y;
    float X2 = e.WiimoteState.IRState.IRSensors[1].RawPosition.X;
    float Y3 = e.WiimoteState.IRState.IRSensors[2].RawPosition.Y;
    float X3 = e.WiimoteState.IRState.IRSensors[2].RawPosition.X;

    // Viertes Infrarotsignal deaktivieren
    e.WiimoteState.IRState.IRSensors[3].Found = false;

    // Update-State Methode aufrufen
    UpdateState(e);


    // Aufruf der Methoden zur Berechnung der Beschleunigungen der IR-Signale
    berechne_A1_y();
    berechne_A2_y();
    berechne_A3_y();

    // Sample auslösen mit Infrarot 1

    // Überprüfen ob:
    // - Beschleunigung größer als Schwellenwert ist
    // - Beschleunigung Maximum nicht überschreitet
    if (A1_y > A_y_Thres && A_y_Max > A1_y)
    {

        // Velocity bestimmen
        float velocity1 = (A1_y - A_y_Thres) * 127 / 4;

        // Velocity als von float in integer Variable überführen
        int veloc1 = (int)velocity1;
```

```
        // Einbeziehen der Velocity Track Bar der GUI
        if (veloc1 < Velocity_min)
            veloc1 = Velocity_min;

        // obere Begrenzung der Velocity
        if (veloc1 > 127)
            veloc1 = 127;

        // Auswahl des richtigen Feldes über Positionsdaten
        if (X1 < 350)
            if (Y1 < 380)
                note = A;
            else
                note = D;
        else
            if (X1 < 700)
                if (Y1 < 380)
                    note = B;
                else
                    note = E;
            else
                if (Y1 < 380)
                    note = C;
                else
                    note = F;


        // Überprüfen der Filtervariable ob Befehl kürzlich ausgelöst worden ist
        if (peak_filter_Y1 == true)
        {
            // MIDI-Befehl senden mit soeben bestimmter Velocity und Tonhöhe
            synth.PlayNote(note, veloc1);

            // Filter Variable aktivieren
            peak_filter_Y1 = false;

            // Feedback Variable auf true setzen
            feedback = true;

        }

    }

    // Sample auslösen mit Infrarot 2
    // für weitere Kommentare siehe "Sample auslösen mit Infrarot 1"
    if (A2_y > A_y_Thres && A_y_Max > A2_y)
    {
        float velocity2 = (A2_y - A_y_Thres) * 127 / 4;

        int veloc2 = (int)velocity2;

        if (veloc2 < Velocity_min)
            veloc2 = Velocity_min;

        if (veloc2 > 127)
            veloc2 = 127;

        if (X2 < 350)
            if (Y2 < 380)
                note = A;
            else
                note = D;
        else
```

```
            if (X2 < 700)
                if (Y2 < 380)
                    note = B;
                else
                    note = E;
            else
                if (Y2 < 380)
                    note = C;
                else
                    note = F;

        if (peak_filter_Y2 == true)
        {
            synth.PlayNote(note, veloc2);
            peak_filter_Y2 = false;
            feedback = true;
        }
    }

    // Sample auslösen mit Infrarot 3
    // für weitere Kommentare siehe "Sample auslösen mit Infrarot 1"
    if (A3_y > A_y_Thres && A_y_Max > A3_y)
    {
        float velocity3 = (A3_y - A_y_Thres) * 127 / 4;

        int veloc3 = (int)velocity3;

        if (veloc3 < Velocity_min)
            veloc3 = Velocity_min;

        if (veloc3 > 127)
            veloc3 = 127;

        if (X3 < 350)
            if (Y3 < 380)
                note = A;
            else
                note = D;
        else
            if (X3 < 700)
                if (Y3 < 380)
                    note = B;
                else
                    note = E;
            else
                if (Y3 < 380)
                    note = C;
                else
                    note = F;


        if (peak_filter_Y3 == true)
        {
            synth.PlayNote(note, veloc3);
            peak_filter_Y3 = false;
            feedback = true;
        }
    }
}
```

```
// Methode zum Auswählen der Switch-Befehle, die ständig von neuem durchlaufen
// wird, sobald die Wiimote Daten sendet
void wm_WiimoteChanged_Switch(object sender, WiimoteChangedEventArgs e)
{

    // Positionsdaten der IR-Signale der Wiimote abrufen
    float Y1 = e.WiimoteState.IRState.IRSensors[0].RawPosition.Y;
    float X1 = e.WiimoteState.IRState.IRSensors[0].RawPosition.X;
    float Y2 = e.WiimoteState.IRState.IRSensors[1].RawPosition.Y;
    float X2 = e.WiimoteState.IRState.IRSensors[1].RawPosition.X;
    float Y3 = e.WiimoteState.IRState.IRSensors[2].RawPosition.Y;
    float X3 = e.WiimoteState.IRState.IRSensors[2].RawPosition.X;

    // Viertes Infrarotsignal deaktivieren
    e.WiimoteState.IRState.IRSensors[3].Found = false;

    // Aufruf der Methoden zur Berechnung der Geschwindigkeiten in x-Richtung
    berechne_V1_x();
    berechne_V2_x();
    berechne_V3_x();

    // Aufruf der Methoden zur Bestimmung der
    // Positionsdifferenzen bezüglich der x-Achse
    berechne_X_Gap12();
    berechne_X_Gap13();
    berechne_X_Gap23();

    // Überprüfen ob Switch-Befehl kürzlich ausgelöst wurde
    if (peak_filter_X == true)

        // Auslösen des Switch-Forward Befehls mit IR 2 & 3

        // Überprüfen ob:
        // - Betätigungs-IR-Signale Mindestgeschwindigkeit
        //   in POSITIVE x-Richtung besitzen
        // - Switch Befehl zufällig ausgelöst wird
        // - Betätigungs-IR-Signale ähnliche x-Koordinaten besitzen
        // - Auswahl-IR-Signal sich im Arbeitsfeld befindet
        if ((V2_x > V_x_Thres_plus && V3_x > V_x_Thres_plus) &&
            (V2_x < V_x_Max && V3_x < V_x_Max) &&
            (X_Gap23 < X_Gap_Max) && (X1 < 1023 ))

            // - Auswahl des richtigen Feldes über Positionsdaten des Auswahl-
            //   IR-Signals und Aufruf der passenden Methode inklusive
            // - Filter Variable auf False setzen und Filter Timer starten
            if (X1 < 350)
                if (Y1 < 380)
                {
                    A_Switch_fwd();
                    peak_filter_X = false;
                    //timer4.Start();
                }
                else
              {
                    D_Switch_fwd();
                    peak_filter_X = false;
                    //timer4.Start();
                }
            else
                if (X1 < 700)
                    if (Y1 < 380)
                    {
                        B_Switch_fwd();
```

```
                peak_filter_X = false;
                //timer4.Start();
            }
            else
            {
                E_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }
        else
            if (Y1 < 380)
            {
                C_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }
            else
            {
                F_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }

// Auslösen des Switch-Forward Befehls mit IR 1 & 3
// für weitere Kommentare siehe "Auslösen des Switch-Forward Befehls
// mit IR 2 & 3"
if ((V1_x > V_x_Thres_plus && V3_x > V_x_Thres_plus) &&
    (V1_x < V_x_Max && V3_x < V_x_Max) &&
    (X_Gap13 < X_Gap_Max) && X2 < 1023)

    if (X2 < 350)
        if (Y2 < 380)
        {
            A_Switch_fwd();
            peak_filter_X = false;
            //timer4.Start();
        }
        else
        {
            D_Switch_fwd();
            peak_filter_X = false;
            //timer4.Start();
        }
    else
        if (X2 < 700)
            if (Y2 < 380)
            {
                B_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }
            else
            {
                E_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }
        else
            if (Y2 < 380)
            {
                C_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
```

```
            }
            else
            {
                F_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }

// Auslösen des Switch-Forward Befehls mit IR 1 & 2
// für weitere Kommentare siehe "Auslösen des Switch-Forward Befehls
// mit IR 2 & 3"
if ((V1_x > V_x_Thres_plus && V2_x > V_x_Thres_plus) &&
    (V1_x < V_x_Max && V2_x < V_x_Max) &&
    (X_Gap12 < X_Gap_Max) && X3 < 1023)

    if (X3 < 350)
        if (Y3 < 380)
        {
            A_Switch_fwd();
            peak_filter_X = false;
            //timer4.Start();
        }
        else
        {
            D_Switch_fwd();
            peak_filter_X = false;
            //timer4.Start();
        }
    else
        if (X3 < 700)
            if (Y3 < 380)
            {
                B_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }
            else
            {
                E_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }
        else
            if (Y3 < 380)
            {
                C_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }
            else
            {
                F_Switch_fwd();
                peak_filter_X = false;
                //timer4.Start();
            }


// Auslösen des Switch-Back Befehls mit IR 2 & 3

// Überprüfen ob:
// - Betätigungs-IR-Signale Mindestgeschwindigkeit
//   in NEGATIVE x-Richtung besitzen
// - Switch Befehl zufällig ausgelöst wird
```

```
// - Betätigungs-IR-Signale ähnliche x-Koordinaten besitzen
// - Auswahl-IR-Signal sich im Arbeitsfeld befindet
if ((V2_x < V_x_Thres_minus && V3_x < V_x_Thres_minus) &&
    (V2_x > V_x_Min && V3_x > V_x_Min) &&
    (X_Gap23 < X_Gap_Max) && (X1 < 1023))

    // Auswahl des richtigen Feldes über Positionsdaten des Auswahl-
    // IR-Signals und Aufruf der passenden Methode inklusive
    // Filter Variable auf False setzen und Filter Timer starten
    if (X1 < 350)
        if (Y1 < 380)
        {
            A_Switch_back();
            peak_filter_X = false;
            //timer4.Start();
        }
        else
        {
            D_Switch_back();
            peak_filter_X = false;
            //timer4.Start();
        }
    else
        if (X1 < 700)
            if (Y1 < 380)
            {
                B_Switch_back();
                peak_filter_X = false;
                //timer4.Start();
            }
            else
            {
                E_Switch_back();
                peak_filter_X = false;
                //timer4.Start();
            }
        else
            if (Y1 < 380)
            {
                C_Switch_back();
                peak_filter_X = false;
                //timer4.Start();
            }
            else
            {
                F_Switch_back();
                peak_filter_X = false;
                //timer4.Start();
            }

// Auslösen des Switch-Back Befehls mit IR 1 & 3
// für weitere Kommentare siehe "Auslösen des Switch-Back Befehls
// mit IR 2 & 3"
if ((V1_x < V_x_Thres_minus && V3_x < V_x_Thres_minus) &&
    (V1_x > V_x_Min && V3_x > V_x_Min) &&
    (X_Gap13 < X_Gap_Max) && X2 < 1023)

    if (X2 < 350)
        if (Y2 < 380)
        {
            A_Switch_back();
            peak_filter_X = false;
            //timer4.Start();
```

```
            }
            else
            {
                D_Switch_back();
                peak_filter_X = false;
                //timer4.Start();
            }
        else
            if (X2 < 700)
                if (Y2 < 380)
                {
                    B_Switch_back();
                    peak_filter_X = false;
                    //timer4.Start();
                }
                else
                {
                    E_Switch_back();
                    peak_filter_X = false;
                    //timer4.Start();
                }
            else
                if (Y2 < 380)
                {
                    C_Switch_back();
                    peak_filter_X = false;
                    //timer4.Start();
                }
                else
                {
                    F_Switch_back();
                    peak_filter_X = false;
                    //timer4.Start();
                }

    // Auslösen des Switch-Back Befehls mit IR 1 & 2
    // für weitere Kommentare siehe "Auslösen des Switch-Back Befehls
    // mit IR 2 & 3"
    if ((V2_x < V_x_Thres_minus && V1_x < V_x_Thres_minus) &&
         (V2_x > V_x_Min && V1_x > V_x_Min) &&
         (X_Gap12 < X_Gap_Max) && X3 < 1023)

        if (X3 < 350)
            if (Y3 < 380)
            {
                A_Switch_back();
                peak_filter_X = false;
                //timer4.Start();
            }
            else
            {
                D_Switch_back();
                peak_filter_X = false;
                //timer4.Start();
            }
        else
            if (X3 < 700)
                if (Y3 < 380)
                {
                    B_Switch_back();
                    peak_filter_X = false;
                    //timer4.Start();
                }
```

```
                    else
                    {
                        E_Switch_back();
                        peak_filter_X = false;
                        //timer4.Start();
                    }
                else
                    if (Y3 < 380)
                    {
                        C_Switch_back();
                        peak_filter_X = false;
                        //timer4.Start();
                    }
                    else
                    {
                        F_Switch_back();
                        peak_filter_X = false;
                        //timer4.Start();
                    }
        }



        // UpdateState Methode, die ständig von neuem durchlaufen wird,
        // sobald die Wiimote Daten sendet
        public void UpdateState(WiimoteChangedEventArgs args)
        {
            // Ruft die UpdateWiimoteChanged Methode durch die asynchrone Ausführung
            // des UpdateWiimoteStateDelegate-Delegaten auf
            BeginInvoke(new UpdateWiimoteStateDelegate(UpdateWiimoteChanged), args);
        }

        // UpdateWiimoteChanged-Methode vermittelt zur UpdateIR-Methode
        // und zeichnet Rahmen und Gitternetz der GUI
        private void UpdateWiimoteChanged(WiimoteChangedEventArgs args)
        {
            // Instanzieren der WiimoteState Klasse aus der WiimoteLib
            WiimoteState ws = args.WiimoteState;

            // Im Designer erstellte Picture Box IRBox als Bitmap b definieren
            IRBox.Image = b;

            // Zeichenoberfläche komplett Weiss definieren
            g.Clear(Color.White);


            // Rand zeichnen
            g.DrawLine(new Pen(Color.Black, 2), 1, 1, 1, 432);
            g.DrawLine(new Pen(Color.Black, 2), 575, 0, 575, 432);
            g.DrawLine(new Pen(Color.Black, 2), 1, 1, 576, 1);
            g.DrawLine(new Pen(Color.Black, 2), 0, 431, 576, 431);


            // Feldabgrenzung zeichnen
            g.DrawLine(new Pen(Color.Black, 2), 192, 0, 192, 432);
            g.DrawLine(new Pen(Color.Black, 2), 384, 0, 384, 432);
            g.DrawLine(new Pen(Color.Black, 2), 0, 216, 576, 216);


            // Visuelles Feedback anzeigen, falls Sample-Auslösung erfolgt
            if (feedback == true)
                if (note == A)
                    g.FillRectangle(new SolidBrush(Color.Green), 0, 0, 192, 216);
```

```
            else if (note == B)
                g.FillRectangle(new SolidBrush(Color.Green), 192, 0, 192, 216);
            else if (note == C)
                g.FillRectangle(new SolidBrush(Color.Green), 386, 0, 192, 216);
            else if (note == D)
                g.FillRectangle(new SolidBrush(Color.Green), 0, 216, 192, 216);
            else if (note == E)
                g.FillRectangle(new SolidBrush(Color.Green), 192, 216, 192, 216);
            else if (note == F)
                g.FillRectangle(new SolidBrush(Color.Green), 386, 216, 192, 216);


        ////// Notenwerte der einzelnen Felder auf die GUI projizieren:

        // Integer-Variable A des Notenwerts im Feld A
        // in String-Variable TextA konvertieren
        TextA = Convert.ToString(A);
        // String Variable TextA als Schriftzug TextNoteA der GUI definieren
        TextNoteA.Text = TextA;

        // folgende Vorgänge analog zum obigen für die anderen Felder
        TextB = Convert.ToString(B);
        TextNoteB.Text = TextB;
        TextC = Convert.ToString(C);
        TextNoteC.Text = TextC;
        TextD = Convert.ToString(D);
        TextNoteD.Text = TextD;
        TextE = Convert.ToString(E);
        TextNoteE.Text = TextE;
        TextF = Convert.ToString(F);
        TextNoteF.Text = TextF;

        // UpdateIR Methode aufrufen mit Übergabe der nötigen Variablen des
        // Infrarotsensors und der definierten Farbe
        UpdateIR(ws.IRState.IRSensors[0], Color.Red);
        UpdateIR(ws.IRState.IRSensors[1], Color.RoyalBlue);
        UpdateIR(ws.IRState.IRSensors[2], Color.DarkGreen);
        UpdateIR(ws.IRState.IRSensors[3], Color.Yellow);

    }

    // Methode zum Darstellen der IR-Punkte
    //
    private void UpdateIR(IRSensor irSensor, Color color)
    {
        // Überprüft ob IR-Signale von der Kamera erkannt werden
        if (irSensor.Found)
        {
            // IR-Punkte zeichnen
            g.FillRectangle(new SolidBrush(color),
            (int)((irSensor.RawPosition.X) * 9 / 16),
            (int)((irSensor.RawPosition.Y) * 9 / 16), 4, 4);
        }

    }


  /// Timer-Methoden zur Verhinderung von Doppeltauslösen der Samples

    // Timer-Methode für die Bewegung von IR 1
    void timer1_Tick(object sender, EventArgs e)
    {
        // Filter-Variable für deaktivieren
```

```
            peak_filter_Y1 = true;
        }
        // Timer-Methode für die Bewegung von IR 2
        void timer2_Tick(object sender, EventArgs e)
        {
            // Filter-Variable deaktivieren
            peak_filter_Y2 = true;
        }
        // Timer-Methode für die Bewegung von IR 3
        void timer3_Tick(object sender, EventArgs e)
        {
            // Filter-Variable deaktivieren
            peak_filter_Y3 = true;
        }

        // Timer-Methode zur Verhinderung von Doppeltauslösen des Switch Befehls
        void timer4_Tick(object sender, EventArgs e)
        {
            // Filter-Variable deaktivieren
            peak_filter_X = true;
        }

        void timer5_Tick(object sender, EventArgs e)
        {
            feedback = false;
        }


////////// GUI Methoden

        // Methode des Optionsmenüs
        private void mIDISetupToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // neue Instaz der MidiOptions-Klasse instanzieren
            MidiOptions options = new MidiOptions();

            // Festlegen der Midi-Optionen im Menü
            options.OutDeviceId = synth.OutputDeviceId;
            options.ProgramId = options.ChannelId;
            options.ChannelId = options.ProgramId;

            // Übertragen der MIDI-Optionen auf das Synth-Objekts durch
            // Drücken des OK-Buttons im Midioptionsmenü
            if (options.ShowDialog() == DialogResult.OK)
            {
                synth.OutputDeviceId = options.OutDeviceId;
                synth.MidiChannel = options.ChannelId;
                synth.MidiProgram = options.ProgramId;
            }
        }

        //// Methode, die über die GUI den Mindest-Velocity-Wert steuert und anzeigt
        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            // Integer Variable Velocity_min als Wert des trackBar1 Elements
            // der GUI festlegen
            Velocity_min = trackBar1.Value;

            //// Anzeigen des Mindest-Velocity-Werts

            // Integer-Variable Velocity_min in
            // String-Variable Text_Vel_min konvertieren
            Text_Vel_min = Convert.ToString(Velocity_min);
```

```
        // String Variable Text_Vel_min als Schriftzug auf der GUI definieren
        label_Vel_min.Text = Text_Vel_min;
    }

    // Methode, die über die GUI die Sensitivity steuert
    private void numericUpDown1_ValueChanged(object sender, EventArgs e)
    {
        // Definiert die Sensitivity Variable als Wert des
        // NumericUpDown1-Elements der GUI
        Sens = numericUpDown1.Value;
    }

//// Methoden die die Begrenzungen der Notenwerte der Felder auf der GUI steuern

    // Notenwertbegrenzung vor Feld A (siehe Feld A & B)
    private void numericUpDown2_ValueChanged(object sender, EventArgs e)
    {
        if (numericUpDown2.Value >= numericUpDown3.Value)
        {
            numericUpDown2.Value = numericUpDown3.Value - 1;
        }
        if (A < (int)numericUpDown2.Value)
        {
            A = (int)numericUpDown2.Value;
        }
    }
    // Notenwertbegrenzung zwischen Feld A & B
    private void numericUpDown3_ValueChanged(object sender, EventArgs e)
    {
        // Notenwertbegrenzung darf nicht kleiner oder gleich
        // voriger Notenwertbegrenzung sein
        if (numericUpDown3.Value <= numericUpDown2.Value)
        {
            numericUpDown3.Value = numericUpDown2.Value + 1;
        }

        // Notenwertbegrenzung darf nicht größer oder gleich
        // nächster Notenwertbegrenzung sein
        if (numericUpDown3.Value >= numericUpDown4.Value)
        {
            numericUpDown3.Value = numericUpDown4.Value - 1;
        }

        // Notenwert des linken Feldes darf nicht größer oder gleich
        // der Notenwertbegrenzung
        if (A >= (int)numericUpDown3.Value)
        {
            A = (int)numericUpDown3.Value - 1;
        }

        // Notenwert des rechten Feldes darf nicht kleiner sein als
        // Notenwertgrenze
        if (B < (int)numericUpDown3.Value)
        {
            B = (int)numericUpDown3.Value;
        }
    }
    // Notenwertbegrenzung zwischen Feld B & C (siehe Feld A & B)
    private void numericUpDown4_ValueChanged(object sender, EventArgs e)
    {
        if (numericUpDown4.Value <= numericUpDown3.Value)
        {
            numericUpDown4.Value = numericUpDown3.Value + 1;
```

```
        }
        if (numericUpDown4.Value >= numericUpDown5.Value)
        {
            numericUpDown4.Value = numericUpDown5.Value - 1;
        }
        if (B >= (int)numericUpDown4.Value)
        {
            B = (int)numericUpDown4.Value - 1;
        }
        if (C < (int)numericUpDown4.Value)
        {
            C = (int)numericUpDown4.Value;
        }
    }
    // Notenwertbegrenzung zwischen Feld C & D (oben rechts) (siehe Feld A & B)
    private void numericUpDown5_ValueChanged(object sender, EventArgs e)
    {
        numericUpDown6.Value = numericUpDown5.Value;

        if (numericUpDown5.Value <= numericUpDown4.Value)
        {
            numericUpDown5.Value = numericUpDown4.Value + 1;
        }
        if (numericUpDown5.Value >= numericUpDown7.Value)
        {
            numericUpDown5.Value = numericUpDown7.Value - 1;
        }
        if (C >= (int)numericUpDown5.Value)
        {
            C = (int)numericUpDown5.Value - 1;
        }
        if (D < (int)numericUpDown5.Value)
        {
            D = (int)numericUpDown5.Value;
        }


    }
    // Notenwertbegrenzung zwischen Feld C & D (unten links) (siehe Feld A & B)
    private void numericUpDown6_ValueChanged(object sender, EventArgs e)
    {
        numericUpDown5.Value = numericUpDown6.Value;

        if (numericUpDown6.Value <= numericUpDown4.Value)
        {
            numericUpDown6.Value = numericUpDown4.Value + 1;
        }
        if (numericUpDown6.Value >= numericUpDown7.Value)
        {
            numericUpDown6.Value = numericUpDown7.Value - 1;
        }
        if (C >= (int)numericUpDown6.Value)
        {
            C = (int)numericUpDown6.Value - 1;
        }
        if (D < (int)numericUpDown6.Value)
        {
            D = (int)numericUpDown6.Value;
        }


    }
    // Notenwertbegrenzung zwischen Feld D & E (siehe Feld A & B)
```

```
        private void numericUpDown7_ValueChanged(object sender, EventArgs e)
        {
            if (numericUpDown7.Value <= numericUpDown6.Value)
            {
                numericUpDown7.Value = numericUpDown6.Value + 1;
            }
            if (numericUpDown7.Value >= numericUpDown8.Value)
            {
                numericUpDown7.Value = numericUpDown8.Value - 1;
            }
            if (C >= (int)numericUpDown7.Value)
            {
                C = (int)numericUpDown7.Value - 1;
            }
            if (D < (int)numericUpDown7.Value)
            {
                D = (int)numericUpDown7.Value;
            }
        }
        // Notenwertbegrenzung zwischen Feld E & F (siehe Feld A & B)
        private void numericUpDown8_ValueChanged(object sender, EventArgs e)
        {
            if (numericUpDown8.Value <= numericUpDown7.Value)
            {
                numericUpDown8.Value = numericUpDown7.Value + 1;
            }
            if (numericUpDown8.Value >= numericUpDown9.Value)
            {
                numericUpDown8.Value = numericUpDown9.Value - 1;
            }
            if (C >= (int)numericUpDown8.Value)
            {
                C = (int)numericUpDown8.Value - 1;
            }
            if (D < (int)numericUpDown8.Value)
            {
                D = (int)numericUpDown8.Value;
            }
        }
        // Notenwertbegrenzung nach Feld F (siehe Feld A & B)
        private void numericUpDown9_ValueChanged(object sender, EventArgs e)
        {
            if (numericUpDown9.Value < numericUpDown8.Value)
            {
                numericUpDown9.Value = numericUpDown8.Value;
            }

            if (F > (int)numericUpDown9.Value)
            {
                F = (int)numericUpDown9.Value;
            }
        }

        //// Methode, die beim Schließen des Fensters ausgeführt wird
        private void Form1_FormClosing(Object sender, FormClosingEventArgs e)
        {
            // Trennt die Verbindung mit der Wiimote
            wm.Disconnect();
        }

    }
}
```

9.2.2. Class MidiOptions

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Sanford.Multimedia.Midi;
using Sanford.Multimedia;

namespace WindowsFormsApplication1
{
    public partial class MidiOptions : Form
    {
        int outDeviceId = 0;
        int channelId = 9;
        int programId = 0;

        internal MidiOptions()
        {
            InitializeComponent();

            PopulateOutputDevices();
            PopulateGMInstruments();

        }

        private void PopulateOutputDevices()
        {
            outputDeviceCombo.Items.Clear();
            if (OutputDevice.DeviceCount > 0)
            {
                for (int i = 0; i < OutputDevice.DeviceCount; i++)
                {
                    outputDeviceCombo.Items.Add
                        (OutputDevice.GetDeviceCapabilities(i).name);
                }
                outputDeviceCombo.SelectedIndex = outDeviceId;
            }
        }

        private void PopulateGMInstruments()
        {
            gmMidiInstrument.Items.Clear();
            for (int i = 0; i < 127; i++)
            {
                gmMidiInstrument.Items.Add((GeneralMidiInstrument)i);
            }
        }

        internal int OutDeviceId
        {
            get { return outDeviceId; }
            set
            {
                outDeviceId = value;
                outputDeviceCombo.SelectedIndex = outDeviceId;
            }
        }
```

```
        internal int ChannelId
        {
            get { return channelId; }
            set
            {
                channelId = value;
                midiChannel.Value = value + 1;
            }
        }

        internal int ProgramId
        {
            get { return programId; }
            set
            {
                programId = value;
                midiInstrument.Value = value + 1;
                gmMidiInstrument.SelectedIndex = value;
            }
        }

        private void outputDeviceCombo_SelectedIndexChanged
        (object sender, EventArgs e)
        {
            outDeviceId = outputDeviceCombo.SelectedIndex;
        }

        private void midiChannel_ValueChanged(object sender, EventArgs e)
        {
            channelId = (int)midiChannel.Value - 1;
        }

        private void midiInstrument_ValueChanged(object sender, EventArgs e)
        {
            programId = (int)midiInstrument.Value - 1;
            gmMidiInstrument.SelectedIndex = programId;
        }

        private void gmMidiInstrument_SelectedIndexChanged(object sender, EventArgs e)
        {
            programId = gmMidiInstrument.SelectedIndex;
            midiInstrument.Value = programId + 1;
        }

    }
}
```

Abbildungsverzeichnis

Abkürzungsverzeichnis

CIL	Common Intermediate Language
CLR	Common Language Runtime
GUI	Graphical User Interface
IR	Infrarot
LED	Leuchtdiode
MIDI	Musical Instrument Digital Interface
MPC	Midi Production Center

Literaturverzeichnis

Bruder, Ralph. Schlick, Christopher und Luczak, Holger. Arbeitswissenschaft. Heidelberg: Springer, 2010

Chadabe, Joel. Electric Sound. The Past and Promise of Electronic Music. New York: Prentice Hall,1997

Claussen, Jan Torge. „Interfacing Audio" – Das Mensch-Maschine-Verhältnis in der digitalen Musikproduktion. Lüneburg, 2006

DIN EN ISO 9241-110 Ergonomie der-Mensch System-Interaktion – Grundsätze der Dialoggestaltung. Berlin: Beuth, 2010

DUDEN – Das Fremdwörterbuch. 9. Auflage. Mannheim: Dudenredaktion, 2007

Engh, Marcel. Artist & Repertoire (A&R). Eine markentheoretische Betrachtung. Musikrezeption, Musikdistribution und Musikproduktion. Der Wandel des Wertschöpfungsnetzwerks in der Musikwirtschaft. Hg. Gerhard Gensch, Eva Maria Stöckler, Peter Tschmuck. Wiesbaden: Gabler, 2008

Franz, Benjamin und Schader, Nils. Einsatzmöglichkeiten der Gestensteuerung bei PC-Anwendungen unter Verwendung von Beschleunigungs- und Infrarotsensoren. Darmstadt, 2008

Gehlen, Arnold. „Die Seele im technischen Zeitalter. Sozialpsychologische Probleme in der industriellen Gesellschaft". Rowohls deutsche Enzyklopädie. Hg. Prof. Ernesto Grassi. Hamburg: Rowohlt, 1957

Grossmann, Rolf. „Sechs Thesen zu musikalischen Interfaces". Weltbilder Bildmedien. Computergestütze Visionen. Interface 2. Hg. Klaus Peter. Hamburg: Dencker, 1995

Harenberg, Michael. „Virtuelle Instrumente zwischen Simulation und (De)Konstruktion". Soundcultures. Über elektronische und digitale Musik. Hg. Marcus S. Kleiner, Achim Szepanski. Frankfurt am Main: Suhrkamp, 2003

Hassenzahl , Marc. Eckoldt, Kai. Thielsch, T. Meinald. User Experience und Experience Design - Konzepte und Herausforderungen. Hg. H. Brau, S. Diefenbach, M. Hassenzahl, K. Kohler, F. Koller, M. Peissner, K. Petrovic, M. Thielsch, D. Ullrich, D. Zimmermann. 2009

Hatscher, Michael. Joy of use – Determinanten der Freude bei der Software-Nutzung: Mensch & Computer 2001: 1. Fachübergreifende Konferenz. Hg. H. Oberquelle, R. Oppermann, J. Krause. Stuttgart: B.G.Teubner, 2001, s.445-446

Hubig, Christoh. Die Kunst des Möglichen II. Ethik als provisorische Moral. Bielefeld: Transcript, 2007

Hubig, Christoph und Jelden, Eva. „Werkuege, Maschinen und Systeme – Leben in der Technik". Funkkolleg Technik. Weinheim: Deutsches Institut für Fernstudienforschung, 1994

Luhmann, Niklas. Die Gesellschaft der Gesellschaft. Frankfurt am Main: Suhrkamp, 1997

Maier, Moritz. Logic Profi Guide. Know-how für bessere Musikproduktion. Bergkirchen: PPVMedien, 2008

Nordmann, Alfred. Technikphilosophie zur Einführung. Hamburg: Junius, 2008

Norman, Donald A.. "Introduction to This Special Section on Beauty, Goodness, and Usability". Human-Computer Interaction Journal 19 (2004): 311-318

Sauter, Martin. Grundkurs Mobile Kommunikationssysteme. Von UMTS und HDSPA, GSM und GPRS zu Wireless LAN und Bluetooth Piconetzen. Wiesbaden: Viewig, 2008

Smudits, Alfred. Soziologie der Musikproduktion. Musikrezeption, Musikdistribution und Musikproduktion. Der Wandel des Wertschöpfungsnetzwerks in der Musikwirtschaft. Hg. Gerhard Gensch, Eva Maria Stöckler, Peter Tschmuck. Wiesbaden: Gabler, 2008

Spinas P., Waeber D., Strohm O.. Kriterien benutzerorientierter Dialoggestaltung und partizipative Softwareentwicklung: eine Literaturaufarbeitung: Benutzerorientierte Softwareentwicklung und Schnittstellengestaltung. (Projektbericht Nr1 Institut für Arbeitspsychologie, ETH Zürich). Hg. P. Spinas, M. Rauterberg, O. Strohm, D. Waeber, E. Ulich, 1990

Stange-Elbe, Kai und Bronner,Kai. Musikinstrumentenindustrie im digitalen Paradigmenwechsel. Musikrezeption, Musikdistribution und Musikproduktion. Der Wandel des Wertschöpfungsnetzwerks in der Musikwirtschaft. Hg. Gerhard Gensch, Eva Maria Stöckler, Peter Tschmuck. Wiesbaden: Gabler, 2008

Stapelkamp, Torsten. Interacion- und Interfacedesign. Web-, Game-, Produkt- und Servicedesign. Usability und Interface als Corporate Identity. Heidelberg: Springer, 2010

Weber, Alexander. „Tablets im Studio“. Beat 12 (2010)

Internetquellen

Bitkom.org. Computerausstattung in Haushalten. URL: http://www.bitkom.org/de/markt_statistik/46261_38547.aspx, letzter Zugriff: 16.05.11

Bruder, Ralph. Vorlesung Gestaltung von Mensch-Maschine-Schnittstellen. Kapitel 1: Einführung, 2010. URL: http://www.arbeitswissenschaft.de/website/teaching/archive/auslegung_v_729/de/docs/mms_2010_kapitel_01.pdf, letzter Zugriff: 16.05.11

Codeplex.com. Project Hosting for Open Source Software. URL: http://www.codeplex.com/, letzter Zugriff: 16.05.11

De-bug.de. DIY: Ziggybox. URL: http://de-bug.de/musiktechnik/archives/4696.html, letzter Zugriff: 16.05.11

De.statista.com. Anteil der Beschäftigten in der EU, die bei der Arbeit einen PC verwenden. URL: http://de.statista.com/statistik/daten/studie/162953/umfrage/anteil-der-europaeischen-beschaeftigten-die-bei-der-arbeit-einen-pc-verwenden/, letzter Zugriff: 16.05.11

Fahrenbach, Ludwig. Ändert die Wissenschaft ständig ihre Meinung?. URL: http://www.dgphil2008.de/programm/sektionen/abstract/fahrbach.html, letzter Zugriff: 16.05.11

Gunnerson, Eric. C#. Die neue Sprache für Microsofts .NET.Plattform URL: http://openbook.galileocomputing.de/csharp/index.htm, letzter Zugriff: 16.05.11

HDM Stuttgart. Musikproduktion im digitalen Wandel, 2008. URL: http://www.hdm-stuttgart.de/~curdt/Musikproduktion im Digitalen Wandel 1.pdf, letzter Zugriff: 16.05.11

Hexler.net. TouchOSC. URL: http://hexler.net/software/touchosc, letzter Zugriff: 16.05.11

Ingelmann, Anja. Usability: elektronische Geräte einfach benutzbar machen, 2010. URL: http://www.echo-online.de/freizeit/multimedia/digitales/Usability-Elektronische-Geraete-einfach-benutzbar-machen;art592,1371413 letzter Zugriff: 16.05.11

Jorda, Sergi. Kaltenbrunner, Martin. Geiger, Günther, Bencina, Ross. The Reactable, 2005. URL: http://mtg.upf.edu/files/publications/9d0455-ICMC2005-JordaKaltenbrunnerGeigerBencina.pdf, letzter Zugriff: 16.05.11

Kinecthacks.net. Windows 7 Mouse With Both Left and Right Click. URL: http://kinecthacks.net/windows-7-mouse-with-both-left-and-right-click/, letzter Zugriff: 16.05.11

König, Christina. Übung 1 - Gestaltung Mensch-Maschine-Schnittstellen. User Interface und benutzergerechte Gestaltung, 2010. URL: http://www.arbeitswissenschaft.de/website/teaching/archive/auslegung_v_730/de/docs/uebung_01_wahrnehmung.pdf, .letzter Zugriff: 16.05.11

König, Christina und Röbig, Andreas. Übung 4 - Gestaltung Mensch-Maschine-Schnittstellen. User Interface und benutzergerechte Gestaltung, 2010. URL: http://www.arbeitswissenschaft.de/website/teaching/archive/auslegung_v_730/de/docs/uebung_04_din9241.pdf, letzter Zugriff: 16.05.11

Kühnel, Andreas. Visual C# 2010. Das umfassende Handbuch. URL: http://openbook.galileocomputing.de/visual_csharp_2010/index.htm#_top, letzter Zugriff: 16.05.11

Lee, Johnny Chung. URL: http://johnnylee.net/projects/wii/, letzter Zugriff: 16.05.11

Midi.org. MIDI Manufacturers Association. URL: http://www.midi.org/index.php, letzter Zugriff: 16.05.11

Mike.verdone.ca. Wii 2 Midi. URL: http://mike.verdone.ca/wiitomidi/, letzter Zugriff: 16.05.11

Moore, Ken. Wii Theremin – How it works, 2008. URL: http://kenmooredesign.blogspot.com/2008/11/wii-theremin.html, letzter Zugriff: 16.05.11

Mulb.org. Verwendung der Wii Remote am PC, 2008. URL: http://www.mulb.org/node/320, letzter Zugriff: 16.05.11

Mutools.com. MU.LAB Free 3. URL: http://www.mutools.com/downloads.html, letzter Zugriff: 16.05.11

Native-Instruments.com. Battery 3. URL: http://www.native-instruments.com/#/de/products/producer/battery-3/, letzter Zugriff: 16.05.11

Nerds.de. LoopBe1 – Free Virtual MIDI Driver. URL: http://www.nerds.de/en/loopbe1.html, letzter Zugriff: 16.05.11

Nintendo.com. Wii Music. URL: http://www.nintendo.com/games/detail/Fe0_TFVoa6RbkoZq_GoIDaRTgOzVAOID, letzter Zugriff: 16.05.11

Ostertag, Bob. Why Computer Music Sucks. URL: http://bobostertag.com/writings-articles-computer-music-sucks.htm, letzter Zugriff: 16.05.11

Peek, Brian. Managed Library for Nintendo's Wiimote. 2009. URL: http://wiimotelib.codeplex.com/, letzter Zugriff: 16.05.11

Pixil.info. Making music with iPad. URL: http://pixil.info/downloads/making-music-with-ipad/, letzter Zugriff: 16.05.11

Röbig, Sinja und Wagner, Torsten. Übung 5 - Gestaltung Mensch-Maschine-Schnittstellen. Usability, 2010. URL: http://www.arbeitswissenschaft.de/website/teaching/archive/auslegung_v_730/de/docs/uebung_05_usability_ss_2010_sr.pdf, letzter Zugriff: 16.05.11

Reactable.com. URL: http://www.reactable.com/, letzter Zugriff: 16.05.11

Reactable.com Tutorials. URL: http://www.reactable.com/products/live/tutorials/, letzter Zugriff: 16.05.11

Sanford, Leslie. C# MIDI Toolkit. 2007. URL: http://www.codeproject.com/KB/audio-video/MIDIToolkit.aspx#Messages, letzter Zugriff: 16.05.11

Sequenzer.de. Synthesizer Grundlagen/ Basics. URL: http://www.sequencer.de/synthaudio/synthesizer-grundlagen.html, letzter Zugriff: 16.05.11

Synthopia.com. Wiimote Synthesizer. URL: http://www.synthtopia.com/content/2008/01/11/wiimote-synthesizer/, letzter Zugriff: 16.05.11

The-breaks.com. Rap Sample FAQ. URL: http://www.the-breaks.com/search.php?term=Funky+Drummer&type=4, letzter Zugriff: 16.05.11

Weiser, Mark. The computer for the 21st century, 1991. URL: http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html, letzter Zugriff: 16.05.11

Wiibrew.org. URL: http://wiibrew.org/wiki/Wiimote, letzter Zugriff: 16.05.11

Wiidrumsynth.codeplex.com. Wii Drum Synth. URL: http://wiidrumsynth.codeplex.com/, letzter Zugriff: 16.05.11

Wiimote Based Applications. .NET-based Wiimote Applications. URL: http://www.brianpeek.com/blog/pages/net-based-wiimote-applications.aspx, letzter Zugriff: 16.05.11

Wikipedia.org. Intuition. URL: http://de.wikipedia.org/wiki/Intuition, letzter Zugriff: 16.05.11

Worldusabilityday.org. URL: http://www.worldusabilityday.org/, letzter Zugriff: 16.05.11

Youtube.com. WiiMusic Drums. URL: http://www.youtube.com/watch?v=Xa_cTiMUjJQ, letzter Zugriff: 16.05.11